IBM WebSphere Application Server for IBM i, Version 8.5

Overview

IBM

Note

Before using this information, be sure to read the general information under “Notices” on page 1155.

Compilation date: June 1, 2012

Contents

How to send your comments

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

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

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

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

Print sections directly from the information center navigation

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

To print a section of the information center navigation:

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

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

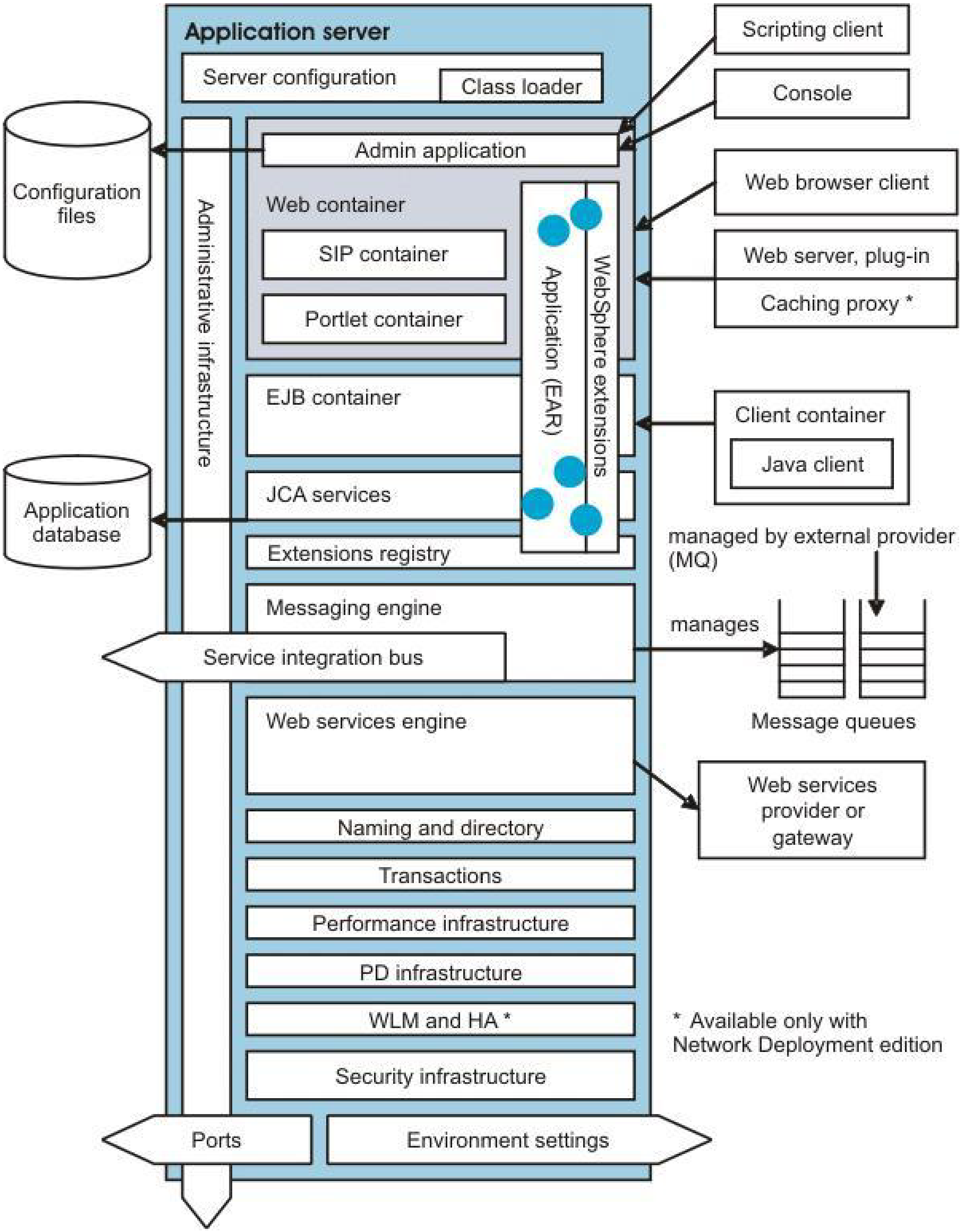
Chapter 1. Learn about WebSphere applications: Overview and new features

Use the **Learn about WebSphere applications** section as a starting point to study the programing model, encompassing the many parts used in and by various application types supported by the application server.

The programming model for applications deployed on this product has the following aspects.

- Java specifications and other open standards for developing applications
- WebSphere® programming model extensions to enhance application functionality
- Containers and services in the application server, used by deployed applications, and which sometimes can be extended

The diagram shows a single application server installation. The parts pertaining to the programming model are discussed here. Other parts comprise the product architecture, independent of the various application types outlined by the programming model. See “WebSphere Application Server architecture” on page 15.



Java EE application components

The product supports application components that conform to Java Platform, Enterprise Edition (Java EE) specifications.

Web applications run in the web container

The web container is the part of the application server in which web application components run. Web applications are comprised of one or more related servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files that you can manage as a unit. Combined, they perform a business logic function.

The web container processes servlets, JSP files, and other types of server-side includes. Each application server runtime has one logical web container, which can be modified, but not created or removed. Each web container provides the following.

Web container transport chains

Requests are directed to the web container using the web container inbound transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP requests, and a web container channel over which requests for servlets and JSP files are sent to the web container for processing.

Servlet processing

When handling servlets, the web container creates a request object and a response object, then invokes the servlet service method. The web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

Servlets can perform such tasks as supporting dynamic web page content, providing database access, serving multiple clients at one time, and filtering data.

JSP files enable the separation of the HTML code from the business logic in web pages. IBM® extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming.

HTML and other static content processing

Requests for HTML and other static content that are directed to the web container are served by the web container inbound chain. However, in most cases, using an external web server and web server plug-in as a front end to the web container is more appropriate for a production environment.

Session management

Support is provided for the javax.servlet.http.HttpSession interface as described in the Servlet application programming interface (API) specification.

An *HTTP session* is a series of requests to a servlet, originating from the same user at the same browser. Sessions allow applications running in a web container to keep track of individual users. For example, many web applications allow users to dynamically collect data as they move through the site, based on a series of selections on pages they visit. Where the user goes next, or what the site displays next, might depend on what the user has chosen previously from the site. To maintain this data, the application stores it in a “session”.

SIP applications and their container

SIP applications are Java programs that use at least one Session Initiation Protocol (SIP) servlet. SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging.

Portlet applications and their container

Portlet applications are special reusable Java servlets that appear as defined regions on portal pages. Portlets provide access to many different applications, services, and web content.

EJB applications run in the EJB container

The EJB container provides all of the runtime services needed to deploy and manage enterprise beans. It is a server process that handles requests for both session and entity beans.

Enterprise beans are Java components that typically implement the business logic of Java EE applications, as well as accessing data. The enterprise beans, packaged in EJB modules, installed in an application server do not communicate directly with the server. Instead, the EJB container is an interface between EJB components and the application server. Together, the container and the server provide the enterprise bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative perspective, the container handles data access for the contained beans. A single container can host more than one EJB Java archive (JAR) file.

Client applications and other types of clients

In a client-server environment, clients communicate with applications running on the server. *Client applications* or *application clients* generally refers to clients implemented according to a particular set of Java specifications, and which run in the client container of a Java EE-compliant application server. Other clients in the WebSphere Application Server environment include clients implemented as web applications (*web clients*), clients of web services programs (*web services clients*), and clients of the product systems administration (*administrative clients*).

Client applications and their container

The client container is installed separately from the application server, on the client machine. It enables the client to run applications in an EJB-compatible Java EE environment. The diagram shows a Java client running in the client container.

This product provides a convenient launchClient tool for starting the application client, along with its client container runtime.

Depending on the source of technical information, client applications sometimes are called application clients. In this documentation, the two terms are synonymous.

Web clients, known also as web browser clients

The diagram shows a web browser client, which can be known simply as a web client, making a request to the web container of the application server. A web client or web browser client runs in a web browser, and typically is a web application.

Web services clients

Web services clients are yet another kind of client that might exist in your application serving environment. The diagram does not depict a web services client. The web services information includes information about this type of client.

Administrative clients

The diagram shows two kinds of administrative clients: a scripting client and the administrative console that is the graphical user interface (GUI) for administering this product. Both are accessing parts of the systems administration infrastructure. In the sense that they are basically the same for whatever kind of applications you are deploying on the server, administrative clients are part of the product architecture. However, because many of these clients are programs you create, they are discussed as part of the programming model for completeness.

Web services

Web services

The diagram shows the web services engine, part of the web services support in the application server runtime. Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a service-oriented architecture (SOA), which supports the connecting or sharing of resources and data in a flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

The product acts as both a web services provider and as a requestor. As a provider, it hosts web services that are published for use by clients. As a requester, it hosts applications that invoke web services from other locations. The diagram shows the web services engine in this capacity, contacting a web services provider or gateway.

SCA composites

Service Component Architecture (SCA)

SCA composites consist of components that implement business functions in the form of services.

Data access, messaging, and Java EE resources

Data access resources

Connection management for access to enterprise information systems (EIS) in the application server is based on the Java EE Connector Architecture (JCA) specification. The diagram shows JCA services helping an application to access a database in which the application retrieves and persists data.

The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts between the application server and EIS.

The Connection Manager (not shown) in the application server pools and manages connections. It is capable of managing connections obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions specification.

JDBC resources (JDBC providers and data sources) are a type of *Java EE resource* used by applications to access data. Although data access is a broader subject than that of JDBC resources, this information often groups data access under the heading of Java EE resources for simplicity.

JCA resource adapters are another type of Java EE resource used by applications. The JCA defines the standard architecture for connecting the Java EE platform to heterogeneous EIS. Imagine an ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between Java EE application servers or clients and an EIS. To use a resource adapter, install the resource adapter code and create configurations that use that adapter. The product provides a predefined relational resource adapter for your use.

Messaging resources and messaging engines

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Applications can use message-driven beans to automatically retrieve messages from JMS destinations and JCA endpoints without explicitly polling for messages.

For inbound non-JMS requests, message-driven beans use a Java EE Connector Architecture (JCA) 1.5 resource adapter written for that purpose. For JMS messaging, message-driven beans can use a JCA-based messaging provider such as the default messaging provider that is part of the product.

The messaging engine supports the following types of message providers.

Default messaging provider (service integration bus)

The default messaging provider uses the service integration bus for transport. The default message provider provides point-to-point functions, as well as publish and subscribe functions. Within this provider, you define JMS connection factories and destinations that correspond to service integration bus destinations.

WebSphere MQ provider
You can use WebSphere MQ as the external JMS provider. The application server provides the JMS client classes and administration interface, while WebSphere MQ provides the queue-based messaging system.

Generic JMS provider
You can use another messaging provider as long as it implements the ASF component of the JMS 1.0.2 specification. JMS resources for this provider cannot be configured using the administrative console.

transition: Version 6 replaces the Version 5 concept of a JMS server with a messaging engine built into the application server, offering the various kinds of providers mentioned previously. The Version 5 messaging provider is offered for configuring resources for use with Version 5 embedded messaging. You also can use the Version 5 default messaging provider with a service integration bus.

EJB 2.1 introduces an ActivationSpec for connecting message-driven beans to destinations. For compatibility with Version 5, you still can configure JMS message-driven beans (EJB 2.0) against a listener port. For those message-driven beans, the message listener service provides a listener manager that controls and monitors one or more JMS listeners, each of which monitors a JMS destination on behalf of a deployed message-driven bean.

Service integration bus

The service integration bus provides a unified communication infrastructure for messaging and service-oriented applications. The service integration bus is a JMS provider that provides reliable message transport and uses intermediary logic to adapt message flow intelligently into the network. It supports the attachment of web services requestors and providers. Its capabilities are fully integrated into product architecture, including the security, system administration, monitoring, and problem determination subsystems.

The service integration bus is often referred to as just a bus. When used to host JMS applications, it is often referred to as a messaging bus. It consists of the following parts (not shown at this level of detail in the diagram).

Bus members
Application servers added to the bus.

Messaging engine
The component that manages bus resources. It provides a connection point for clients to produce or from where to consume messages.

Destinations
The place within the bus to which applications attach to exchange messages. Destinations can represent web services endpoints, messaging point-to-point queues, or messaging publish and subscribe topics. Destinations are created on a bus and hosted on a messaging engine.

Message store
Each messaging engine uses a set of tables in a supported data store (such as a JDBC database) to hold information such as messages, subscription information, and transaction states.

Through the service integration bus web services enablement, you can:

- Make an internal service that is already available at a service destination available as a web service.
- Make an external web service available at a service destination.
- Use the web services gateway to map an existing service, either an internal service or an external web service, to a new web service that appears to be provided by the gateway.

Mail, URLs, and other Java EE resources

The following kinds of Java EE resources are used by applications deployed on a J2EE-compliant application server.

- JDBC resources and other technology for data access (previously discussed)
- JCA resource adapters (previously discussed)
- JMS resources and other messaging support (previously discussed)
- JavaMail support, for applications to send Internet mail

  The **JavaMail APIs** provide a platform and protocol-independent framework for building Java-based mail client applications. The APIs require service providers, known as protocol providers, to interact with mail servers that run on the appropriate protocols.

  A mail provider encapsulates a collection of protocol providers, including Simple Mail Transfer Protocol (SMTP) for sending mail; Post Office Protocol (POP) for receiving mail; and Internet Message Access Protocol (IMAP) as another option for receiving mail. To use another protocol, you must install the appropriate service provider for the protocol.

  JavaMail requires not only service providers, but also the JavaBeans Activation Framework (JAF), as the underlying framework to handle complex data types that are not plain text, such as Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.
- URLs, for describing logical locations

  URL providers implement the functionality for a particular URL protocol, such as HTTP, enabling communication between the application and a URL resource that is served by a particular protocol. A default URL provider is included for use by any URL resource with protocols based on the supported Java Platform, Standard Edition (Java SE) specification, such as HTTP, FTP, or File. You also can plug in your own URL providers that implement additional protocols.
- Resource environment entries, for mapping logical names to physical names

  The java:comp/env environment provides a single mechanism by which both the JNDI name space objects and local application environment objects can be looked up. The product provides numerous local environment entries by default.

  The Java EE specification also provides a mechanism for defining customer environment entries by defining entries in the standard deployment descriptor of an application. The Java EE specification uses the following methods to separate the definition of the resource environment entry from the application.

  - Requiring the application server to provide a mechanism for defining separate administrative objects that encapsulate a resource environment entry. The administrative objects are accessible using JNDI in the application server local name space (java:comp/env).
  - Specifying the administrative object's JNDI lookup name and expected returned object type. This specification is performed in the aforementioned resource environment entry in the deployment descriptor.

  The product supports the use of resource environment entries with the following administrative concepts.

  - A *resource environment entry* defines the binding target (JNDI name), factory class, and return object type (via the link to a referenceable) of the resource environment entry.
  - A *referenceable* defines the class name of the factory that returns object instances implementing a Java interface.
  - A *resource environment provider* groups together the referenceable, resource environment entries and any required custom properties.

Security

Security programming model and infrastructure

The product provides security infrastructure and mechanisms to protect sensitive Java EE resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

Security infrastructure and mechanisms protect Java Platform, Enterprise Edition (Java EE) resources and administrative resources, addressing your enterprise security requirements. In turn, the security infrastructure of this product works with the existing security infrastructure of your multiple-tier enterprise computing framework. Based on open architecture, the product provides many plug-in points to integrate with enterprise software components to provide end-to-end security.

The security infrastructure involves both a programming model and elements of the product architecture that are independent of the application type.

Additional services for use by applications

Naming and directory

Each application server provides a naming service that in turn provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register resources hosted on the application server. The JNDI implementation is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model used by application developers. CosNaming provides the server-side implementation and is where its name space is actually stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming, and interacts with the CosNaming server on behalf of the client.

Clients of the application server use the naming architecture to obtain references to objects related to those applications. The objects are bound into a mostly hierarchical structure called the name space. It consists of a set of name bindings, each one of which is a name relative to a specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.

This product provides the following naming and directory features.

- Distributed name space, for additional scalability
- Transient and persistent partitions, for binding at various scopes
- Federated name space structure across multiple servers
- Configured bindings for defining bindings bound by the system at server startup
- Support for CORBA Interoperable Naming Service (INS) object URLs

Note that with the addition of virtual member manager to provide federated repository support for product security, the product now offers more extensive and sophisticated identity management capabilities than ever before, especially in combination with other WebSphere and Tivoli® products.

Object Request Broker (ORB)

The product uses an ORB to manage interaction between client applications and server applications, as well as among product components. An ORB uses IIOP to enable clients to make requests and receive requests from servers in a network distributed environment.

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency.

Although not shown in the diagram, one place in which the ORB comes into play is where the client container is contacting the EJB container on behalf of a Java client.

Transactions

Part of the application server is the transaction service. The product provides advanced transactional capabilities to help application developers avoid custom coding. It provides support for the many challenges related to integrating existing software assets with a Java EE environment. These measures include ActivitySessions (described below).

Applications running on the server can use transactions to coordinate multiple updates to resources as one unit of work such that all or none of the updates are made permanent. Transactions are started and ended by applications or the container in which the applications are deployed.

The application server is a transaction manager that supports coordination of resource managers and participates in distributed global transactions with other compliant transaction managers.

The server can be configured to interact with databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction support is not required.

How applications use transactions depends on the type of application, for example:

- A session bean either can manage its transactions itself, or delegate the management of transactions to the container.
- Entity beans use container-managed transactions.
- Web components, such as servlets, use bean-managed transactions.

The product handles transactions with the following components.

- A transaction manager supports the enlistment of recoverable XAResources and ensures each resource is driven to a consistent outcome, either at the end of a transaction, or after a failure and restart of the application server.
- A container manages the enlistment of XAResources on behalf of deployed applications when it performs updates to transactional resource managers such as databases. Optionally, the container can control the demarcation of transactions for EJB applications that have enterprise beans configured for container-managed transactions.
- An API handles bean-managed enterprise beans and servlets, allowing such application components to control the demarcation of their own transactions.

WebSphere extensions

WebSphere programming model extensions are the programming model benefits you gain by purchasing this product. They represent leading edge technology to enhance application capability and performance, and make programming and deployment faster and more productive.

In addition, your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis. The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers who build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. For more information about this feature, see Application extension registry.

The various WebSphere programming model extensions, and the corresponding application services that support them in the application server runtime, can be considered in three groups: Business Object Model extensions, Business Process Model extensions, and extensions for producing Next Generation Applications.

Extensions pertaining to the Business Object Model

Business object model extensions operate with business objects, such as enterprise bean (EJB) applications.

Application profiling

Application profiling is a WebSphere extension for defining strategies to dynamically control concurrency, prefetch, and read-ahead.

Application profiling and access intent provide a flexible method to fine-tune application performance for enterprise beans without impacting source code. Different enterprise beans, and even different methods in one enterprise bean, can have their own intent to access resources. Profiling the components based on their access intent increases performance in the application server runtime.

Dynamic query

Dynamic query is a WebSphere programming extension for unprecedented application flexibility. It lets you dynamically build and submit queries that select, sort, join, and perform calculations on application data at runtime. Dynamic Query service provides the ability to pass in and process EJB query language queries at runtime, eliminating the need to hard-code required queries into deployment descriptors during application development.

Dynamic query improves enterprise beans by enabling the client to run custom queries on EJB components during runtime. Until now, EJB lookups and field mappings were implemented at development time and required further development or reassembly in order to be changed.

Dynamic cache

The dynamic cache service improves performance by caching the output of servlets, commands, and JSP files. This service within the application server intercepts calls to cacheable objects and either stores the output of the object or serves the content of the object from the dynamic cache.

Because Java EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create opportunity for significant gains in server response time, throughput, and scalability.

Features include cache replication among clusters, cache disk offload, Edge side include caching, and external caching - the ability to control caches outside of the application server, such as that of your Web server.

Extensions pertaining to the Business Process Model

Business process model extensions provide process, workflow functionality, and services for the application server. Use them in conjunction with business integration capabilities.

ActivitySessions

ActivitySessions are a WebSphere extension for reducing the complexity of dealing with commitment rules and limitations associated with one-phase commit resources.

ActivitySessions provide the ability to extend the scope of multiple local transactions, and to group them. This enables them to be committed based on deployment criteria or through explicit program logic.

Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Extensions for creating next generation applications

Next generation extentions can be used in applications that need the specific extensions. These enable next generation development by leveraging the latest innovations that build on today's Java EE standards. This provides greater control over application development, execution, and performance than was ever possible before.

Asynchronous beans

Asynchronous beans offer performance enhancements for resource-intensive tasks by enabling single tasks to run as multiple tasks. Asynchronous scheduling facilities can also be used to process parallel processing requests in “batch mode” at a designated time. The product provides full support for asynchronous execution and invocation of threads and components within the application server. The application server provides execution and security context for the components, making them an integral part of the application.

Startup beans

Startup beans allow the automatic execution of business logic when the application server starts or stops. For example, they might be used to pre-fill application-specific caches, initialize application-level connection pools, or perform other application-specific initialization and termination procedures.

Object pools

Object pools provide an effective means of improving application performance at runtime, by allowing multiple instances of objects to be reused. This reuse reduces the overhead associated with instantiating, initializing, and garbage-collecting the objects. Creating an object pool allows an application to obtain an instance of a Java object and return the instance to the pool when it has finished using it.

Internationalization

The internationalization service is a WebSphere extension for improving developer productivity. It allows you to automatically recognize the time zone and location information of the calling client, so that your application can act appropriately. The technology enables you to deliver each user, around the world, the right date and time information, the appropriate currencies and languages, and the correct date and decimal formats.

Scheduler

The scheduler service is a WebSphere programming extension responsible for starting actions at specific times or intervals. It helps minimize IT costs and increase application speed and responsiveness by maximizing utilization of existing computing resources. The scheduler service provides the ability to process workloads using parallel processing, set specific transactions as high priority, and schedule less time-sensitive tasks to process during low traffic off-hours.

Work areas

Work areas are a WebSphere extension for improving developer productivity. Work areas provide a capability much like that of “global variables”. They provide a solution for passing and propagating contextual information between application components.

Work areas enable efficient sharing of information across a distributed application. For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it will be available throughout your application, eliminating the need to hand-code a solution or to read and write information to a database.

Guided activities for the administrative console

The topic describes the *guided activities* that are available in the administrative console. Guided activities lead you through common administrative tasks that require you to visit multiple administrative console pages.

Table 1. Quick reference: Accessing the guided activities. The following table gives you the web address for the guided activities in the administrative console.

The guided activities are available from the main page of the administrative console. The page is displayed after you log into the administrative console. To open the console, enter this web address in your web browser: *http://your_fully_qualified_server_name*:9060/ibm/console Depending on your configuration, your web address might differ. Other factors can affect your ability to access the console. See Starting and logging off the administrative console for details, as needed.

Guided activities display each administrative console page that you need to perform a task, surrounded by the following information to help you perform the task successfully.

- An introduction to the task, introducing essential concepts and describing when and why to perform the task
- Other tasks to do before and after performing the task
- The main steps to complete during this task
- Hints and tips to help you avoid and recover from problems
- Links to field descriptions and extended task information in the online documentation

Tutorials

This topic describes how to find tutorials and their accompanying samples, for learning how to accomplish your goals with the product.

IBM Education Assistant tutorials
The IBM Education Assistant site provides education resources that you can use at your convenience.

developerWorks tutorials and training
The Tutorials and Training page of developerWorks provides tutorials and other training resources that you can use at your convenience.

Accessing the samples

The product offers samples that demonstrate common enterprise application tasks. Many samples also provide instructions for deployment and coding examples.

The product provides samples in two ways:

Plants By WebSphere sample installed with the product
If you select to install samples when installing the product and when creating an application server profile, the Plants By WebSphere application is included with the product. The application demonstrates several Java Platform, Enterprise Edition (Java EE) functions, using an online store that specializes in plant and garden tool sales.

See Installing the Plants By WebSphere sample.

Samples downloadable from the Samples, Version 8.5 information center
The product provides component-specific samples that you can download at any time from a download site.

- Available samples

- Downloading samples

Installing the Plants By WebSphere sample

To install the Plants By WebSphere sample, perform the following steps.

1. Install the product.

   Plants By WebSphere sample is installed in the *app_server_root*/`samples` directory. A Plants By WebSphere pre-built enterprise archive named `pbw-ear.ear` is in the `/samples/PlantsByWebSphere/pbw-ear/target` directory.

   Installation instructions are in the `/samples/PlantsByWebSphere/docs` directory.

   You can build or modify the sample source code to support your project. The source code is in a `src` directory.
2. Start the application server.

Available samples

Samples that you can download include, for example, the following materials:

Communications Enabled Applications (CEA) samples
The CEA sample applications provide two main services, telephony access and multi-modal web interaction. Use this collection of sample applications to explore the services and to use as a starting point when developing your own communication enabled applications.

OSGi samples
The OSGi samples help you develop and deploy modular applications that use both Java EE and OSGi technologies.

XML samples
The XML samples demonstrate use of the XML API and supported specifications.

Internationalization service sample
The Internationalization service sample demonstrates how to use the internationalization service in Java EE applications, specifically within servlets and enterprise beans.

Web services samples

These samples demonstrate both Java API for XML-based RPC (JAX-RPC) and Java API for XML Web Services (JAX-WS) web services that use Java Platform, Enterprise Edition (Java EE) beans and JavaBeans components.

The JAX-WS web service samples demonstrate the implementation of one-way and two-way web services that highlight the use of web services standards such as WS-Addressing (WS-A) , WS-Reliable Messaging (WS-RM), and WS-Secure Conversation (WS-SC) and the SOAP Message Transmission Optimization Mechanism (MTOM) technology.

Service Data Objects (SDO) sample
This sample demonstrates data access to a relational database through Service Data Objects (SDO) and Java DataBase Connectivity (JDBC) Mediator technologies.

Downloading samples

You can download samples from the **Samples, Version 8.5** information center.

1. Go to the **Samples, Version 8.5** information center.
2. Determine which samples you want to download.
3. On the **Downloads** tab for the samples that you want, click a **Download Sample** link.
4. In the authentication window, click **OK**.
5. Download the compressed file, or individual sample files, to a directory on your workstation.

You might create the `/samples/sample_type` directory path on your workstation and download the sample files to that directory path.

Many sample compressed files have an `/installableApps` directory that contains deployable prebuilt archives. Other directories contain files such as sample-specific source archives, scripts, and instructions for building deployable archives.

Limitations of the samples

- The samples are for demonstration purposes only.

  The code that is provided is not intended to run in a secured production environment. The samples support Java 2 Security, therefore the samples implement policy-based access control that checks for permissions on protected system resources, such as file I/O.

  The samples also support administrative security.

Additional samples and examples

Samples on developerWorks®

Additional product samples are available on WebSphere developerWorks

Samples in tutorials

Many product tutorials rely on sample code. To find tutorials that demonstrate specific technologies, browse the links in “Tutorials” on page 12.

Examples in the product documentation

The product documentation contains many code snippets and examples. To locate these examples easily, see the developer examples in the **Reference** section of the information center navigation for the product edition that you are using.

IBM i **IBM Telephone Directory**

The IBM Telephone Directory business application is shipped separately from the product. For information about obtaining and using the IBM Telephone Directory application, see IBM Telephone Directory V5.2 in the topic on e-business and web serving in the IBM i information center.

Development and assembly tools

You can use an Integrated Development Environment to develop, assemble, and deploy Java Platform, Enterprise Edition (Java EE) modules for WebSphere Application Server.

The IBM Rational® Application Developer for WebSphere Software product and the IBM WebSphere Application Server Developer Tools for Eclipse product are supported tools for integrated development environments.

This information center refers to the products as the *assembly tools*. However, you can use the products to do more than assemble modules. Use these tools in an integrated development environment to develop, assemble, and deploy Java EE modules.

The Rational Application Developer for WebSphere Software is a more extensive set of tools supporting enterprise development. This workbench has integrated support for WebSphere Application Server Version 6.1 and later. This workbench also supports both the OSGi and Java EE programming models, and contains wizards and visual editors to help you develop Web 2.0, Service Component Architecture (SCA), Java, and Java EE applications. This product contains code quality tools to help you analyze code and improve performance. This product integrates with Rational Team Concert to provide a team-based environment to help developers share information and work collaboratively. The Trial download for Rational Application Developer is available at `http://www.ibm.com/developerworks/downloads/r/rad/`.

IBM WebSphere Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java EE applications to WebSphere Application Server Version 7.0 and 8.x.

This workbench integrates with the application server to help you to quickly deploy and test applications. This product contains wizards and visual editors that support the Java EE programming model.

For documentation on the tools, see “Rational Application Developer documentation.” Topics on application assembly in this information center supplement that documentation.

Important: The assembly tools run on Windows and Linux Intel platforms. Users of WebSphere Application Server on all platforms must assemble their modules using an assembly tool installed on Windows or Linux Intel platforms. To install an assembly tool, follow instructions available with the tool.

WebSphere Application Server architecture

This article introduces the parts of the WebSphere Application Server.

Servers

WebSphere Application Server. An application server is a Java virtual machine (JVM) running user applications. Application servers use Java technology to extend web server capabilities to handle web application requests. An application server makes it possible for a server to generate a dynamic, customized response to a client request. The WebSphere Application Server provides application servers.

IBM i For more introduction, refer to “Introduction: Application servers” on page 994.

For more information, refer to Server collection.

Web servers. In the WebSphere Application Server, an application server works with a web server to handle requests for web applications. The application server and web server communicate using an HTTP plug-in for the web server.

For more information, refer to Implementing a web server plug-in.

Resources

JMS providers. The product supports messaging by providing a range of Java Message Service (JMS) providers that conform to the JMS specifications. There are three main types of JMS provider that can be configured in WebSphere Application Server: The WebSphere Application Server default messaging provider (uses service integration as the provider), the WebSphere MQ messaging provider (uses your WebSphere MQ system as the provider) and 3rd party messaging providers (use another company's product as the provider).

For more information, refer to “Introduction: Messaging resources” on page 1046.

Environment

Environment settings help handle requests among web applications, web containers, and the application server.

Virtual hosts. A virtual host is a configuration enabling a single host to resemble multiple logical hosts. Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number that are used to request the servlet, for example: `hostname:80`. The DNS alias might be the host name and port of a web server that routes to the application server or the actual host name and port on which the application server is listening. Java Platform, Enterprise Edition (Java EE) web modules are mapped to a virtual host at installation time. Web modules that use the same virtual host can dispatch to resources within one another.

For more information, refer to Virtual hosts.

WebSphere variables. Variables are used to control settings and properties relating to the server environment. WebSphere variables are used to configure product path names such as JAVA_HOME, and environmental customization values.

For more information, refer to WebSphere variables.

Shared libraries. Shared libraries are files used by multiple applications. You can define a shared library at the node or server level. You can then associate the library to an application or server in order for the classes represented by the shared library to be loaded in either a server-wide or application-specific class loader.

For more information, refer to Managing shared libraries.

System administration

Administrative console. The administrative console is a graphical interface that provides many features to guide you through deployment and systems administration tasks. Use it to explore available management options.

For more introduction, refer to “Introduction: Administrative console” on page 990.

Scripting client (wsadmin). The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. You can also submit scripting language programs to run. The wsadmin tool is intended for production environments and unattended operations.

For more introduction, refer to “Introduction: Administrative scripting (wsadmin)” on page 990.

Administrative programs (Java Management Extensions). The product supports a Java programming interface for developing administrative programs. All of the administrative tools that are supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification.

For more introduction, refer to “Introduction: Administrative programs” on page 991.

Command line tools. Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

For more introduction, refer to “Introduction: Administrative commands” on page 991.

Configuration files. Product configuration data resides in XML files that are manipulated by the previously mentioned administrative clients.

For more introduction, refer to “Introduction: Administrative configuration data” on page 992.

Monitoring and tuning

Monitoring tools. Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environments. Performance monitoring tools include :

- Performance Monitoring Infrastructure (PMI) for monitoring to understand overall system health. For more information, see Performance Monitoring Infrastructure (PMI).

- Request metrics for monitoring to understand resource usage. For more information, see Why use request metrics?.
- Tivoli Performance Viewer (TPV) for viewing the performance data that you collected. For more information, see Why use Tivoli Performance Viewer?.

Tuning tools. Tuning the product helps you obtain the best performance from your website. Tuning the product involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

For more information, refer to Obtaining advice from the advisors.

Troubleshooting

Diagnostic tools. Diagnostic tools help you isolate the source of problems. Many diagnostic tools are available for this product.

IBM i For more information, refer to Working with troubleshooting tools.

Support and self-help IBM Support can assist in deciphering the output of diagnostic tools. Refer to the WebSphere Application Server Technical Support website for current information on known problems and their resolution. Documents at this site can save you time gathering information that is needed to resolve a problem.

For more information, refer to the WebSphere Application Server Support page.

Three-tier architectures

WebSphere Application Server provides the application logic layer in a three-tier architecture, enabling client components to interact with data resources and legacy applications.

Collectively, three-tier architectures are programming models that enable the distribution of application functionality across three independent systems, typically:

- Client components running on local workstations (tier one)
- Processes running on remote servers (tier two)
- A discrete collection of databases, resource managers, and mainframe applications (tier three)

The following diagram outlines the three tier levels. The tiers are logical . They might or might not be running on the same physical server.

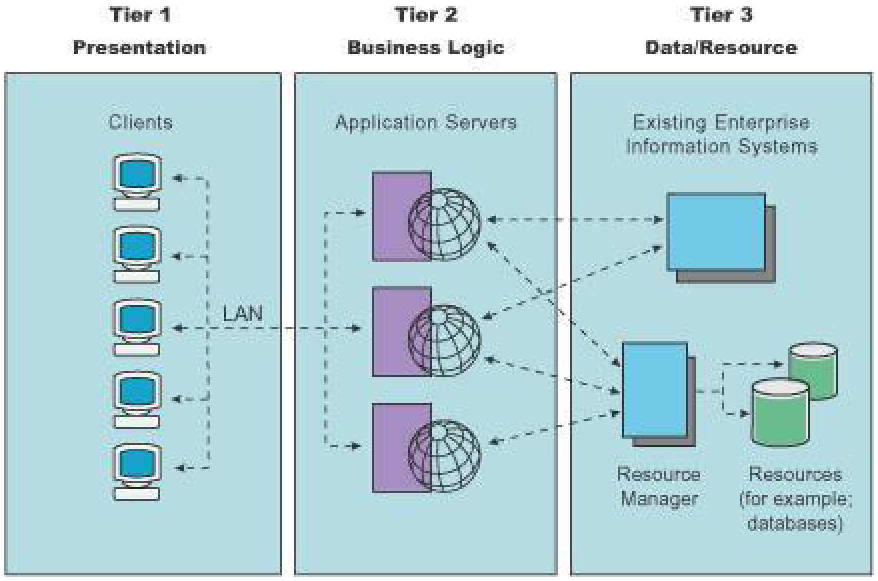


Figure 1. Three tier architecture

First tier. Responsibility for presentation and user interaction resides with the first-tier components. These client components enable the user to interact with the second-tier processes in a secure and intuitive manner. WebSphere Application Server supports several client types. Clients do not access the third-tier services directly. For example, a client component provides a form on which a customer orders products. The client component submits this order to the second-tier processes, which check the product databases and perform tasks that are needed for billing and shipping.

Second tier. The second-tier processes are commonly referred to as the “application logic layer”. These processes manage the business logic of the application, and are permitted access to the third-tier services. The application logic layer is where most of the processing work occurs. Multiple client components can access the second-tier processes simultaneously, so this application logic layer must manage its own transactions.

In the previous example, if several customers attempt to place an order for the same item, of which only one remains, the application logic layer must determine who has the right to that item, update the database to reflect the purchase, and inform the other customers that the item is no longer available. Without an application logic layer, client components access the product database directly. The database is required to manage its own connections, typically locking out a record that is being accessed. A lock can occur when an item is placed into a shopping cart, preventing other customers from considering it for purchase. Separating the second and third tiers reduces the load on the third-tier services, supports more effective connection management, and can improve overall network performance.

Third tier. The third-tier services are protected from direct access by the client components residing within a secure network. Interaction must occur through the second-tier processes.

Communication among tiers. All three tiers must communicate with each other. Open, standard protocols and exposed APIs simplify this communication. You can write client components in any programming language, such as Java or C++. These clients run on any operating system, by speaking with the application logic layer. Databases in the third tier can be of any design, if the application layer can query and manipulate them. The key to this architecture is the application logic layer.

Chapter 2. ActivitySessions

This page provides a starting point for finding information about ActivitySessions, a WebSphere extension for reducing the complexity of commitment rules and limitations that are associated with one-phase commit resources.

Use ActivitySessions to extend the scope and group multiple local transactions. With this capability, you can commit these transactions based on either deployment criteria or through explicit program logic.

The ActivitySession service

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

Support for the ActivitySession service is shown in the following figure:

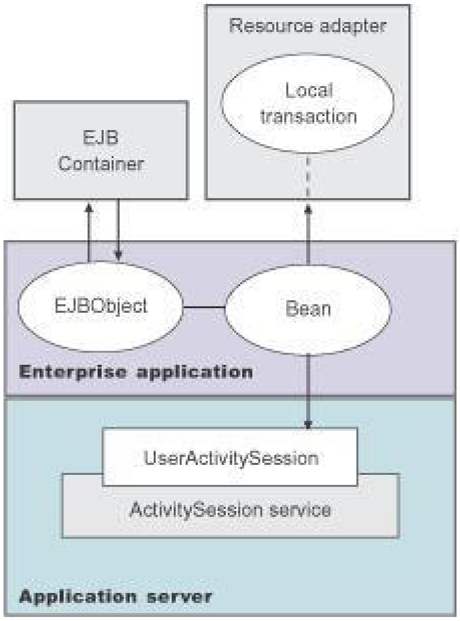


Figure 2. The ActivitySession service. This figure shows the main components of the ActivitySession service in WebSphere Application server. For an overview of these components, see the text that accompanies this figure.

Although the purpose of a global transaction is to coordinate multiple resource managers, enterprise applications often use global transaction context as a “session” context through which to access Enterprise JavaBeans (EJB) instances. An ActivitySession context is such a session context, and can be used in preference to a global transaction in cases where coordination of two-phase commit resource managers is not needed. Further, an ActivitySession can be associated with an HttpSession to extend a “client session” to an HTTP client.

ActivitySession support is available to Web, EJB, and Java platform for enterprise applications client components. EJB components can be divided into beans that exploit container-managed ActivitySessions and beans that use bean-managed ActivitySessions.

The ActivitySession service provides a UserActivitySession application programming interface available to enterprise application components that use bean-managed ActivitySessions for application-managed demarcation of ActivitySession context. The ActivitySession service also provides a system programming interface for container-managed demarcation of ActivitySession context and for container-managed enlistment of one-phase resources (resource manager local transactions (RMLTs)) in such contexts.

The UserActivitySession interface is obtained by a Java Naming and Directory Interface (JNDI) lookup of java:comp/websphere/UserActivitySession. This interface is not available to enterprise beans that use container-managed ActivitySessions, and any attempt by such beans to obtain the interface results in a NotFound exception.

A common scenario is an enterprise application accessing one or more enterprise beans backed by non-transactional (one-phase commit) resources. The application, or its container, uses the UserActivitySession interface to define the demarcation boundaries within which operations against the enterprise beans are grouped and to control whether those grouped operations should be checkpointed or discarded. The business logic of the enterprise beans does not need to use any ActivitySession interfaces. The container into which the enterprise beans are deployed ensures that updates to the underlying one-phase resource managers are coordinated.

The application can checkpoint an ActivitySession to create a new point of consistency within the ActivitySession without ending the ActivitySession. The application can also use a reset operation to return work performed in the ActivitySession back to the last point of consistency. The application can end the ActivitySession with an operation to either checkpoint or reset all resources.

Usage model for using ActivitySessions with HTTP sessions

This topic describes how a Web application that runs in the WebSphere Web container can participate in an ActivitySession context.

If the Web application is designed such that several servlet invocations occur as part of the same logical application, then the servlets can use the HttpSession to preserve state across servlet invocations. The ActivitySession context is one state that can be suspended into the HttpSession and resumed on a future invocation of a servlet that accesses the HttpSession.

An ActivitySession is associated automatically with an HttpSession, so can be used to extend access to the ActivitySession over multiple HTTP invocations, over inclusion or forwarding of servlets, and to support Enterprise JavaBeans (EJB) activation periods that can be determined by the lifecycle of the Web HTTP client. An ActivitySession context stored in an HttpSession can also be used to relate work for the ActivitySession back to a specific Web HTTP client.

The Web container manages ActivitySessions based on deployment descriptor attributes associated with servlets in the Web application module. The two usage models are:

- The Web container starts and ends ActivitySessions.

  The Web application invokes a servlet that has been configured for container control of ActivitySessions.
  - If an HttpSession exists then it has an associated ActivitySession.
  - If an HttpSession does not exist, the servlet can start an HttpSession, which causes an ActivitySession to be started automatically and associated with the HttpSession.

  A servlet cannot start a new HttpSession until an existing HttpSession has been ended. Within an HttpSession, the Web application can invoke other servlets that can use the associated ActivitySession context. When the Web application invokes a servlet that ends the HttpSession, the ActivitySession is ended automatically. This is shown in the following diagram:

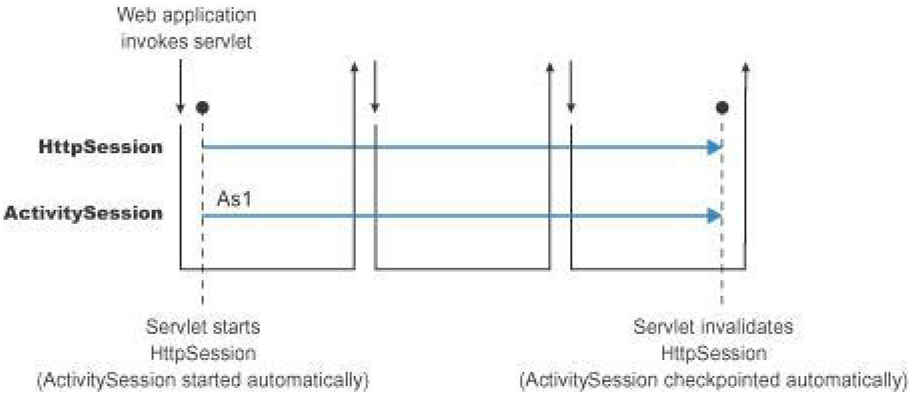

Web application invokes servlet
HttpSession
ActivitySession
As1
Servlet starts HttpSession (ActivitySession started automatically)
Servlet invalidates HttpSession (ActivitySession checkpointed automatically)

Figure 3. Web container control of ActivitySessions. This figure is described in the surrounding text.

- The Web application starts and ends ActivitySessions.

  The Web application invokes a servlet that has been configured for application control of ActivitySesions.

  – If an HttpSession exists and has an associated ActivitySession, the servlet can use or end that ActivitySession context.

  – If an HttpSession does not exist, the servlet can start an HttpSession, but this does not automatically start an ActivitySession.

  – If an HttpSession exists but does not have an associated ActivitySession, the servlet can start a new ActivitySession. This automatically associates the ActivitySession with the HttpSession. The ActivitySession lasts either until the ActivitySession is specifically ended or until the HttpSession is ended.

  The servlet cannot start a new ActivitySession until an existing ActivitySession has been ended. The servlet cannot start a new HttpSession until an existing HttpSession has been ended.

  Within an HttpSession, the Web application can invoke other servlets that can use or end an existing ActivitySession context or, if no ActivitySession exists start a new ActivitySession. When the Web application invokes a servlet that ends the HttpSession, the ActivitySession is ended automatically. This is shown in the following diagram:

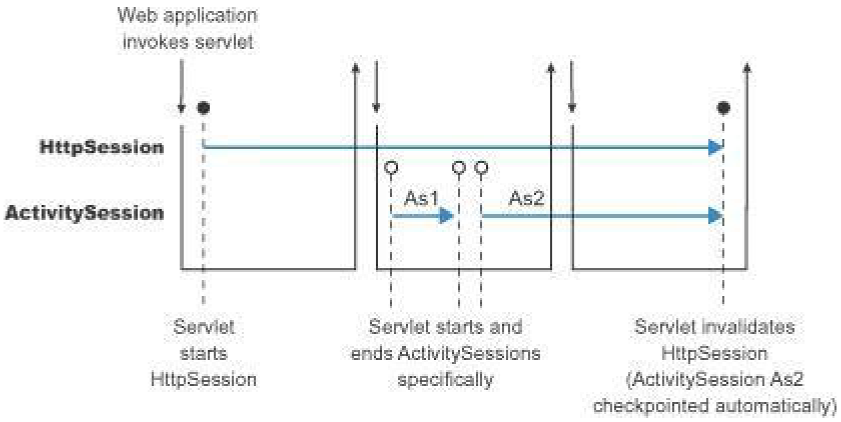

Web application invokes servlet
HttpSession
ActivitySession
As1
As2
Servlet starts HttpSession
Servlet starts and ends ActivitySessions specifically
Servlet invalidates HttpSession (ActivitySession As2 checkpointed automatically)

Figure 4. Web application control of ActivitySessions,. This figure is described in the surrounding text.

A Web application can invoke servlets configured for either usage model.

The following points apply to both usage models:

- To end an HttpSession (and any associated ActivitySession), the Web application must invalidate that session. This causes the ActivitySession to be checkpointed.

- Any downstream enterprise beans activated within the context of an ActivitySession can be held in memory rather than passivated between servlet invocations, because the client effectively becomes the Web HTTP client.
- Web applications can be composed of many servlets, and each servlet in the Web application can be configured with a value for ActivitySessionControl. ActivitySessionControl determines whether the servlet or its container starts any ActivitySessions.
- An ActivitySession context that encapsulates an active transaction context cannot be associated with an HttpSession, because a transaction can hold database locks and should be designed to be shortlived. If an application moves an active transaction to an HttpSession, the transaction is rolled back and the ActivitySession is suspended into the HTTPSession. In general, you should design applications to use ActivitySessions or other constructs as the long-lived entities and ACID transactions as short-duration entities within these.
- Only one ActivitySession can be associated with an HttpSession at any time, for the duration of the ActivitySession. An ActivitySession associated with an HttpSession remains associated for the duration of that ActivitySession, and cannot be replaced with another until the first ActivitySession is completed. The ActivitySession can be accessed by multiple servlets if they have shared access to the HttpSession.
- ActivitySessions are not persistent. If a persistent HttpSession exists longer than the server hosting it, any cached ActivitySession is terminated when the hosting server ends.
- If the HttpSession times out before the associated ActivitySession has ended, then the ActivitySession is reset[1]. This rolls back the ActivitySession resources to the last point of consistency:
  - If the Web application invoked a servlet that has been configured for container control of ActivitySessions, the ActivitySession resources are rolled back completely.
  - If the Web application invoked a servlet that has been configured for application control of ActivitySessions, the ActivitySession resources are rolled back to the last checkpoint taken by the servlet, or completely if no checkpoint has been taken.
- If the ActivitySession times out, it is reset to the last point of consistency (see previous item), then the HttpSession is ended.

ActivitySession and transaction contexts

This topic describes the hierarchical relationship between transaction and ActivitySession contexts. This relationship, defined by the ActivitySession service, requires that any transaction context be either wholly inside or wholly outside an ActivitySession context.

An ActivitySession context is very similar to a transaction context and extends the lifecycle choices for activation of enterprise beans; it can encapsulate one or more transactions. The ActivitySession context is a distributed context that, like the transaction context, can be bean- or container-managed. An ActivitySession context is used mainly by a client to scope the lifecycle of an enterprise bean that it uses either beyond or in the absence of individual transactions started by that client.

ActivitySessions have a lower overhead than transactions and can be used instead of transactions that are only used to scope the lifecycle of a called enterprise bean. For a bean with an activation policy of ActivitySession, the duration of any resource manager local transactions (RMLTs) started by that bean can be bounded by the duration of the ActivitySession instead of the bean method in which the RMLT was started. This provides flexibility and potential for using RMLTs in an enterprise bean beyond the scenarios described in the Enterprise JavaBeans (EJB) specifications. The EJB specifications define that RMLTs need to be completed before the end of the bean method, because the bean method is the only containment boundary for local transactions available in those specifications.

The following rules defines the relationship between transactions and ActivitySessions.

[1] Resetting an ActivitySession causes all the resources involved in the current ActivitySession to be rolled back to the last point of consistency, but allows further work within the ActivitySession. When the reset completes, the thread is associated with the same ActivitySession as it was before the reset was called. The ActivitySession resources remain associated with the ActivitySession although they cannot participate further in the ActivitySession

- The EJB or Web container always uses a local transaction containment (LTC) if there is no global transaction present. An LTC can be method-scoped or ActivitySession-scoped.
- Before a method dispatch, the container ensures that there is always either an LTC or global transaction context, but never both contexts.
- ActivitySessions cannot be nested within each other. Any attempt to start a nested ActivitySession results in a com.ibm.websphere.ActivitySession.NotSupportedException on UserActivitySession.beginSession().
- An ActivitySession can wholly encapsulate one or more global transactions.
- The application can end an ActivitySession with an operation to either checkpoint or reset all resources. The endSession(EndModeCheckpoint) operation checkpoints the work coordinated under the ActivitySession then ends the context. The endSession(EndModeReset) operation resets, to the last point of consistency, the work coordinated under the ActivitySession then ends the context.
- An ActivitySession cannot be encapsulated by a global transaction nor should ActivitySession and global transaction boundaries overlap. Any attempt to start an ActivitySession in the presence of a global transaction context results in a com.ibm.websphere.ActivitySession.NotSupportedException on UserActivitySession.beginSession(). Any attempt to call endSession(EndModeCheckpoint) on an ActivitySession that contains an incomplete global transaction results in a com.ibm.websphere.ActivitySession.ContextPendingException. Neither the global transaction nor the ActivitySession context are affected. If endSession(EndModeReset) is called then the ActivitySession is reset and the global transactions marked rollback_only.
- Each global transaction wholly encapsulated by an ActivitySession is independent of every other global transaction within that ActivitySession. A rollback of one global transaction does not affect any others or the ActivitySession itself.
- ActivitySession and global transaction contexts can coexist with an ActivitySession encapsulating one or more serially-running global transactions.
- EJB home methods cannot participate in an ActivitySession because this situation might cause deadlocks. EJB home methods run in their own independent LTC.

ActivitySession and transaction container policies in combination

This topic provides details about the relationship between the deployment descriptor properties that determine how the container manages ActivitySession boundaries.

If an enterprise bean uses ActivitySessions, how the EJB container manages ActivitySession boundaries when delegating a method invocation depends on both the **ActivitySession kind** and **Container transaction type** deployment descriptor attributes configured for the enterprise bean. The following table lists the relationship between these two properties.

In each row, the final column describes the behavior that the EJB container takes with respect to global transaction and ActivitySession context, based on the following abbreviations:

S*n* An ActivitySession, where *n* indicates the ActivitySession instance.

T*n* A transaction, where *n* indicates the transaction instance.

In every case where the container does not start or leave a global transaction context associated with the thread, it starts (or obtains from the bean instance) a local transaction containment and associates that with the thread. The duration of the local transaction containment is determined by a combination of the local-transaction boundary descriptor (configured as part of the application deployment descriptor, and not shown in the following table) and the presence or not of an ActivitySession context, as described in ActivitySessions and transaction contexts.

The rows highlighted in bold are not allowed.

Table 2. Container behavior for activitysession and transaction policies deployment settings

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Required	Required	None	Start S1, Start T1
		S1	Start T1
		T1	Suspend T1, Start S1, Start T2
		S1, T1	No Action
	Requires new	None	Start S1, Start T1
		S1	Start T1
		T1	Suspend T1, Start S1, Start T2
		S1, T1	Suspend T1, Start T2
	Supports	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	No Action
	Not supported	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	Exception
		S1, T1	No action
	Never	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	Exception

Table 2. Container behavior for activitysession and transaction policies deployment settings (continued)

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Requires new	Required	None	Start S1 + T1
		S1	Suspend S1, Start S2 + T1
		T1	Suspend T1, Start S1 + T2
		S1 + T1	Suspend S1 + T1, Start S2 + T2
	Requires new	None	Start S1 + T1
		S1	Suspend S1, Start S2 + T1
		T1	Suspend T1, Start S1 + T2
		S1 + T1	Suspend S1 + T1, Start S2 + T2
	Supports	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2
	Not supported	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2
	Mandatory	**None**	**Exception**
		S1	**Exception**
		T1	**Exception**
		S1, T1	**Exception**
	Never	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2

Table 2. Container behavior for activitysession and transaction policies deployment settings (continued)

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Supports	Required	None	Start T1
		S1	Start T1
		T1	No Action
		S1, T1	No Action
	Requires new	None	Start T1
		S1	Start T1
		T1	Suspend T1, Start T2
		S1, T1	Suspend T1, Start T2
	Supports	None	No Action
		S1	No Action
		T1	No Action
		S1, T1	No Action
	Not supported	None	No Action
		S1	No Action
		T1	Suspend T1
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1, T1	No Action
	Never	None	No Action
		S1	No Action
		T1	Exception
		S1, T1	Exception

Table 2. Container behavior for activitysession and transaction policies deployment settings (continued)

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Not supported	Required	None	Start T1
		S1	Suspend S1, Start T1
		T1	No Action
		S1, T1	Suspend S1 + T1, Start T2
	Requires new	None	Start T1
		S1	Suspend S1, Start T1
		T1	Suspend T1, Start T2
		S1, T1	Suspend S1 + T1, Start T2
	Supports	None	No Action
		S1	Suspend S1
		T1	No Action
		S1, T1	Suspend S1 + T1
	Not supported	None	No Action
		S1	Suspend S1
		T1	Suspend T1
		S1, T1	Suspend S1 + T1
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Never	None	No Action
		S1	Suspend S1
		T1	Exception
		S1, T1	Suspend S1 + T1

Table 2. Container behavior for activitysession and transaction policies deployment settings (continued)

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Mandatory	Required	None	Exception
		S1	Start T1
		T1	Exception
		S1, T1	No Action
	Requires new	None	Exception
		S1	Start T1
		T1	Exception
		S1, T1	Suspend T1, Start T2
	Supports	None	Exception
		S1	No Action
		T1	Exception
		S1, T1	No Action
	Not supported	None	Exception
		S1	No Action
		T1	Exception
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	Exception
		S1, T1	No Action
	Never	None	Exception
		S1	No Action
		T1	Exception
		S1,T1	Exception

Table 2. Container behavior for activitysession and transaction policies deployment settings (continued)

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Container transaction type)	Received contexts	Container behavior
Never	Required	None	Start T1
		S1	Exception
		T1	No Action
		S1, T1	Exception
	Requires new	None	Start T1
		S1	Exception
		T1	Suspend T1, Start T2
		S1,T1	Exception
	Supports	None	No Action
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Not supported	None	No Action
		S1	Exception
		T1	Suspend T1
		S1,T1	Exception
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Never	None	No Action
		S1	Exception
		T1	Exception
		S1,T1	Exception
Bean managed	Bean managed	None	No Action
		S1	Suspend S1
		T1	Suspend T1
		S1, T1	Suspend S1 + T1

ActivitySession samples

WebSphere Application Server provides some ActivitySession samples.

MasterMind sample

This sample is based on the game MasterMind. It consists of the following components:

- A servlet, configured with the ActivitySession control kind attribute set to Container, that accesses a stateful session bean.
- A stateful session bean, configured with an activation policy of ActivitySession containing transient state data.

The servlet begins an HttpSession at the start of each new game, and ends it at the end of each game; therefore an ActivitySession lasts for the duration of each game. The ActivitySession activation policy stops the bean from being passivated and therefore the transient data remains in memory. This sample demonstrates the association between HttpSession and ActivationSession in the web container, and an ActivitySession-scoped activation policy.

Enterprise application client container and a CMP entity bean backed by a one-phase commit data source

In this sample, the entity bean is configured with the following properties:

- TX_NOT_SUPPORTED
- An ActivitySession container managed policy of REQUIRES
- An LTC boundary of ActivitySession
- An LTC Resolution Control of ContainerAtBoundary

The client accesses the UserActivitySession, begins an ActivitySession, updates two instances of the bean, then ends the ActivitySession. It does this twice using EndModeReset then EndModeCheckpoint. This sample demonstrates the following functionality:

- Client access to the UserActivitySession interface
- Multiple resource manager local transactions (RMLTs) being scoped to the ActivitySession and taking their completion direction automatically from that of the ActivitySession

The entity bean also holds a transient variable that each method call increments (gets and sets for the persistent data). This value is checked before the end of the ActivitySession to show that the same bean instance is used. The client checks for the correct results.

An enterprise application client container and two session beans with different ActivitySession types

This sample consists of an enterprise application client container and the following session beans:

- SLB1, a stateless session bean configured with an ActivitySession Type of Bean.
- SFB2, a stateful session bean configured with ActivitySession Type of Requires, an LTC boundary of ActivitySession, LTC Resolution Contol of APPLICATION, and an LTC Unresolved Action of ROLLBACK.

Both beans are configured with TX_NOTSUPPORTED.

This sample uses the following steps:

1. The client starts SLB1
2. SLB1 accesses the UserActivitySession interface, begins an ActivitySession, then calls a method on SFB2
3. SFB2 accesses the UserActivitySession interface, begins an ActivitySession, calls a method on SFB2
4. SFB2 gets a connection (setAutoCommit false) then uses JDBC to update a single-phase data source.
5. Optionally, SLB1 calls a separate method on SFB2 to finish the work, either committing or rolling back the RMLT.
6. SLB1 then ends the ActivitySession with an EndModeCheckpoint.

This sample demonstrates the following functionality:

- The ActivitySession completion direction is unconnected to the direction of the RMLTs, although the containment of the RMLTs is bound to the ActivitySession.
- The container using the unresolved action when an RMLT is not completed.
- A bean-managed ActivitySessions bean using the UserActivitySession interface.

The sample checks for correct results and reports them back to the client.

ActivitySession service: Resources for learning

Use the links in this topic to find relevant supplemental information about ActivitySessions. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

Programming model and decisions

- The application programming interface (API) reference information.

Programming specifications

- J2EE Activity Service for Extended Transactions
- Java Transaction API (JTA) 1.0.1

Other

- WebSphere Business Integration Server Foundation
- List of IBM WebSphere Redbooks
- WebSphere technical library, including links to white papers

Chapter 3. Application profiling

This page provides a starting point for finding information about application profiling, a WebSphere extension for defining strategies to dynamically control concurrency, prefetch, and read-ahead.

Application profiling and access intent provide a flexible method to fine-tune application performance for enterprise beans without impacting source code. Different enterprise beans, and even different methods in one enterprise bean, can have their own intent to access resources. Profiling the components based on their access intent increases performance in the application server run time.

The application profiling service is not available for Enterprise JavaBeans (EJB) that are contained in a web archive (WAR). As a result, application profiling tasks can not be accessed from an EJB in a WAR.

Application profiling

You can use application profiling to identify particular units of work to the product runtime environment. The run time can tailor its support to the exact requirements of that unit of work.

Application profiling requires accurate knowledge of an application's transactional configuration and the interaction of the application with its persistent state during the course of each transaction.

You can execute the analysis in either closed world or open world mode. A closed-world analysis assumes that all possible clients of the application are included in the analysis and that the resulting analysis is complete and correct. The results of a closed-world analysis report the set of all transactions that can be invoked by a web, JMS, or application client. The results exclude many potential transactions that never execute at run time.

An open-world analysis assumes that not all clients are available for analysis or that the analysis cannot return complete or accurate results. An open-world analysis returns the complete set of possible transactions.

The results of an analysis persist as an application profiling configuration. The assembly tool establishes container managed tasks for servlets, JavaServer Pages (JSP) files, application clients, and Message Driven Beans (MDBs). Application profiles for the tasks are constructed with the appropriate access intent for the entities enlisted in the transaction represented by the task. However, in practice, there are many situations where the tool returns at best incomplete results. Not all applications are amenable to static analysis. Some factory and command patterns make it impossible to determine the call graphs. The tool does not support the analysis of *ActivitySessions*.

You should examine the results of the analysis very carefully. In many cases you must manually modify them to meet the requirements of the application. However, the tool can be an effective starting place for most applications and may offer a complete and quick configuration of application profiles for some applications.

Access intent is the only runtime component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two new concepts in order to achieve this function: *tasks* and *profiles*.

Tasks A task is a configurable name for a unit of work. *Unit of work* in this case means either a transaction or an ActivitySession. The task name is typically assigned declaratively on a J2EE component that can initiate a unit of work. Most commonly, the task is configured on a method of an Enterprise JavaBeans file that is declared either for container-managed transactions or bean-managed transactions. Any unit of work that begins in the scope of a configured task is associated with that task name. A unit of work can only be named when it is initiated, and the name cannot change for the lifetime of that unit of work. A unit of work ignores any subsequent task name configurations at any point after it has begun. The task is used for the duration of its unit of work to identify configured policies specific to that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Profiles

A profile is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a CMR getter, or a dynamic query) requires data to be retrieved from the back end system, the current task associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent. If a request is operating in the absence of a task, the runtime environment uses either a method-level access intent (if any) or a bean-level default access intent.

Note: The application profile configuration is application scope configuration data. If any Enterprise JavaBean (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

Tasks and units of work considerations

The application profiling function works under the unit of work (UOW) concept. UOW in this case means either a transaction or an ActivitySession.

The task name on a method is used only when a UOW is begun, because of that method being invoked. This gives it a more predictable data access pattern based on the active unit of work. To be more specific, this approach ensures that a bean type with only one configured access intent is loaded within a UOW, because a bean is configured with only one access intent within an application profile. This configured access intent for a bean type is determined at assembly time and is enforced by the Application Profile service.

A task name is always associated with a unit of work, and that task name does not change for the duration of that UOW. When a UOW associated with a method is begun because of that method being invoked, if a task name is associated with the method then that task name is used to name the UOW. A task assigned to a unit of work is considered a named UOW.

If a task name is not associated with the method that began the UOW, then a default access intent is used and the UOW is unnamed. A unit of work can only be named when the UOW is begun and that task name remains for the life of the UOW. Furthermore, the task assigned to a UOW can never be changed for the life of that UOW. Any task names associated with a method are ignored if that method does not begin a UOW (either container managed or component managed).

It is not possible to change the task name assigned to a unit of work. However, it is possible that in a call sequence consisting of many different application calls a different task name might need to be used for different calls. In this case it is important for the deployer to begin a new UOW and associate with the UOW the necessary task name. For example, assume you have the following beans: sb1 is a session bean, eb2 and eb3 are container managed persistence (CMP) entity beans. When sb1 is called, a transaction is begun and task 't1' is associated with it. Further assume that sb1 then calls eb2 and eb3. If neither eb2 or eb3 create a unit of work, then these beans execute within the UOW context from sb1 and as such its task name (t1). If eb2 or eb3 need to execute within a task name other than t1, then these beans must define a unit of work and associate with it the appropriate task name.

Note that if an application deployer does not specifically configure a transaction on a method, WebSphere Application Server creates a global transaction by default. This is important because if a task is defined on a method, but a UOW is not specifically configured on that method, the EJB container automatically creates a global transaction on behalf of that method. As such, this task name is associated with the UOW and any application profiles mapped to this task are used.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

Application profiles

An application profile is the set of access intent policies that should be selectively applied for a particular unit of work (a transaction or *ActivitySession*).

Application profiling enables applications to run under different sets of policies depending on the active task under which the application is operating.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Consider an application that centralizes the student records for a school district. These records are frequently accessed by the school district's central office in order to generate reports. The report generation process would be optimized if it held no locks with the back end system, and if the records could be read into memory with as few back end operations as possible. Occasionally, however, the records are updated by the students' instructors. Without the ability to distinguish between transactions, the developer is forced to assume a worst-case scenario and, wishing to use pessimistic concurrency, lock the records for all transactions.

Using the application profiling service, the developer can configure in as many ways as necessary the access intent under which the students' records are loaded. Under one profile, the records can be configured with an exclusive pessimistic update intent, not only locking-out competing transactions but ensuring that the student is not removed from the system before the transaction completes. Under another profile, the records can be configured with an optimistic intent as part of an object graph that is read from the back end system in a single database operation. The task represented by the pessimistic profile receives the strong-locking semantics required for certain transactions, while the task represented by the optimistic profile receives the performance benefits appropriate for other transactions.

Application profiling tasks

Tasks are named units of work. They are the mechanism by which the runtime environment determines which access intent policies to apply when an entity bean's data is loaded from the back end system.

Application profiles enable developers to configure an entity bean with multiple access intent policies; if there are *n* instances of profiles in a given application, each bean can be configured with as many as *n* access intent policies.

A task is associated with a transaction or an ActivitySession at the initiation of the unit of work. The task, which cannot change for the lifetime of the unit of work, is always available anywhere within the scope of that unit of work to apply the access intent policy configured for that particular unit of work.

If an enterprise application is configured to use application profiling in any part of the application, then application profiling is active and method-level access intent configurations are ignored when units of works are associated with known-to-application tasks.

If an entity bean is loaded in a unit of work that is not associated with a task, or is associated with a task that is unassociated with an application profile, the default bean-level access intent or the method-level access intent configuration is applied. If a unit of work is associated with a task that is configured with an application profile, the bean-level access intent configuration within the appropriate application profile is applied.

Note: The application profile configuration is application scope configuration data. If any Enterprise Javabeans (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

For example, consider a school district application that calls through a session bean in order to interact with student records. One method on the session bean allows administrators to modify the students' records; another method supports student requests to view their own records. Without application profiling, the two tasks would operate anonymously and the runtime environment would be unable to distinguish work operating on behalf of one task or the other. To optimize the application, a developer can configure one of the methods on the session bean with the task "updateRecords" and the other method on the session bean with the task "readRecords". When registered with an application profile that has the student bean configured with the appropriate locking access intent, the "updateRecords" task is assured that it is not unnecessarily blocking transactions that need to only read the records. For more information about the relationships between tasks and units of work, see “Tasks and units of work considerations” on page 34.

Tasks can be configured to be managed by the container or to be programmatically established by the application. Container managed tasks can be configured on servlets, JavaServer Pages (JSP) files, application clients, and the methods of Enterprise JavaBeans (EJB). Configured container-managed tasks are only associated with units of work that the container initiates after the task name is set. Application managed tasks can be configured on all J2EE components. In the case of enterprise beans they must be bean managed transactions."

best-practices: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Chapter 4. Asynchronous beans

This page provides a starting point for finding information about asynchronous beans.

Asynchronous beans and asynchronous scheduling facilities offer performance enhancements for resource-intensive tasks by enabling single tasks to run as multiple tasks.

Asynchronous beans

An asynchronous bean is a Java object or enterprise bean that can run asynchronously by a Java Platform, Enterprise Edition (Java EE) application, using the Java EE context of the asynchronous bean creator.

Asynchronous beans can improve performance by enabling a Java EE program to decompose operations into parallel tasks. Asynchronous beans support the construction of stateful, active Java EE applications. These applications address a segment of the application space that Java EE has not previously addressed (that is, advanced applications that require application threading, active agents within a server application, or distributed monitoring capabilities).

Asynchronous beans can run using the Java EE security context of the creator Java EE component. These beans also can run with copies of other Java EE contexts, such as:
- Internationalization context
- Application profiles, which are not supported for Java EE 1.4 applications and deprecated for Java EE 1.3 applications
- Work areas

Asynchronous bean interfaces

Four types of asynchronous beans exist:

Work object

There are two work interfaces that essentially accomplish the same goal. The legacy Asynchronous Beans work interface is com.ibm.websphere.asynchbeans.Work, and the CommonJ work interface is commonj.work.Work. A work object runs parallel to its caller using the work manager `startWork` or `schedule` method (`startWork` for legacy Asynchronous Beans and `schedule` for CommonJ). Applications implement work objects to run code blocks asynchronously.

Timer listener

This interface is an object that implements the commonj\timers\TimerListener interface. Timer listeners are called when a high-speed transient timer expires.

Alarm listener

An alarm listener is an object that implements the com.ibm.websphere.asynchbeans.AlarmListener interface. Alarm listeners are called when a high-speed transient alarm expires.

Event listener

An event listener can implement any interface. An event listener is a lightweight, asynchronous notification mechanism for asynchronous events within a single Java virtual machine (JVM). An event listener typically enables Java EE components within a single application to notify each other about various asynchronous events.

Supporting interfaces

Work manager

Work managers are thread pools that administrators create for Java EE applications. The administrator specifies the properties of the thread pool and a policy that determines which Java EE contexts the asynchronous bean inherits.

CommonJ Work manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a Java EE 1.4 environment, each JNDI lookup of a work manager does not return a new instance of the WorkManager. All the JNDI lookup of work managers within a scope have the same instance.

Timer manager
Timer managers implement the commonj.timers.TimerManager interface, which enables Java EE applications, including servlets, EJB applications, and JCA Resource Adapters, to schedule future timer notifications and receive timer notifications. The timer manager for Application Servers specification provides an application-server supported alternative to using the J2SE `java.util.Timer` class, which is inappropriate for managed environments.

Event source
An event source implements the com.ibm.websphere.asynchbeans.EventSource interface. An event source is a system-provided object that supports a generic, type-safe asynchronous notification server within a single JVM. The event source enables event listener objects, which implement any interface to be registered.

Event source events
Every event source can generate its own events, such as listener count changed. An application can register an event listener object that implements the class com.ibm.websphere.asynchbeans.EventSourceEvents. This action enables the application to catch events such as listeners being added or removed, or a listener throwing an unexpected exception.

Additional interfaces, including alarms and subsystem monitors, are introduced in the Developing asynchronous scopes topic, which discusses some of the advanced applications of asynchronous beans.

Transactions

Every asynchronous bean method is called using its own transaction, much like container-managed transactions in typical enterprise beans. It is very similar to the situation when an Enterprise JavaBeans (EJB) method is called with TX_NOT_SUPPORTED. The runtime starts a local transaction containment before invoking the method. The asynchronous bean method is free to start its own global transaction if this transaction is possible for the calling Java EE component. For example, if an enterprise bean creates the component, the method that creates the asynchronous bean must be TX_BEAN_MANAGED.

When you call an entity bean from within an asynchronous bean, for example, you must have a global transactional context available on the current thread. Because asynchronous bean objects start local transactional contexts, you can encapsulate all entity bean logic in a session bean that has a method marked as TX_REQUIRES or equivalent. This process establishes a global transactional context from which you can access one or more entity bean methods.

If the asynchronous bean method throws an exception, any local transactions are rolled back. If the method returns normally, any incomplete local transactions are completed according to the unresolved action policy configured for the bean. EJB methods can configure this policy using their deployment descriptor. If the asynchronous bean method starts its own global transaction and does not commit this global transaction, the transaction is rolled back when the method returns.

Access to Java EE component metadata

If an asynchronous bean is a Java EE component, such as a session bean, its own metadata is active when a method is called. If an asynchronous bean is a simple Java object, the Java EE component metadata of the creating component is available to the bean. Like its creator, the asynchronous bean can look up the java:comp namespace. This look up enables the bean to access connection factories and enterprise beans, just as it would if it were any other Java EE component. The environment properties of the creating component also are available to the asynchronous bean.

The `java:comp` namespace is identical to the one available for the creating component; the same restrictions apply. For example, if the enterprise bean or servlet has an EJB reference of `java:comp/env/ejb/MyEJB`, this EJB reference is available to the asynchronous bean. In addition, all of the connection factories use the same resource-sharing scope as the creating component.

Connection management

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

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

For code examples that demonstrate both the correct and the incorrect ways to access connections from asynchronous bean methods, refer to the Example: Asynchronous bean connection management topic.

Deferred start of Asynchronous Beans

Asynchronous beans support deferred start by allowing serialization of Java EE service context information. The `WorkWithExecutionContext createWorkWithExecutionContext(Work r)` method on the WorkManager interface will create a snapshot of the Java EE service contexts enabled on the WorkManager. The resulting `WorkWithExecutionContext` object can then be serialized and stored in a database or file. This is useful when it is necessary to store Java EE service contexts such as the current security identity or Locale and later inflate them and run some work within this context. The `WorkWithExecutionContext` object can run using the startWork() and doWork() methods on the WorkManager interface.

All `WorkWithExecutionContext` objects must be deserialized by the same application that serialized it. All EJBs and classes must be present in order for Java to successfully inflate the objects contained within.

Deferred start and security

The asynchronous beans security service context might require Common Secure Interoperability Version 2 (CSIv2) identity assertion to be enabled. Identity assertion is required when a `WorkWithExecutionContext` object is deserialized and run to Java Authentication and Authorization Service (JAAS) subject identity credential assignment. Review the following topics to better understand if you need to enable identity assertion, when using a `WorkWithExecutionContext` object:

- Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocol
- Identity Assertion

There are also issues with interoperating with `WorkWithExecutionContext` objects from different versions of the product. Refer to the Interoperating with asynchronous beans topic.

JPA-related limitations

Use of asynchronous beans within a JPA extended persistence context is not supported.

A JPA extended persistence context is inconsistent with the scheduling and multi-threading capabilities of asynchronous beans and will not be accessible from an asynchronous bean thread.

Likewise, an asynchronous bean should not be created such that it takes a javax.persistence.EntityManager (or subclass) as a parameter since EntityManager instances are not intended to be thread safe.

Work managers

A work manager is a thread pool created for Java Platform, Enterprise Edition (Java EE) applications that use asynchronous beans.

Using the administrative console, an administrator can configure any number of work managers. The administrator specifies the properties of the work manager, including the Java EE context inheritance policy for any asynchronous beans that use the work manager. The administrator binds each work manager to a unique place in Java Naming and Directory Interface (JNDI). You can use work manager objects in any one of the following interfaces:

- Asynchronous beans
- CommonJ work manager (For details, see the CommonJ work manager section in this article.)

The selected type of interface is resolved during the JNDI lookup time. The interface type is the value that you specify in the ResourceRef, rather than the interface type specified in the configuration object. For example, you can have one ResourceRef for each interface per configuration object, and each ResourceRef lookup returns that appropriate type of instance.

The work managers provide a programming model for the Java EE 1.4 applications. For more information, see the Programming model section in this article.

Important: The `javax.resource.spi.work.WorkManager` class is a Java interface to be used by Java EE Connector Architecture (JCA) resource adapters. It is not an actual implementation of the WorkManager which is used by Java EE applications.

When writing a Web or Enterprise JavaBeans (EJB) component that uses asynchronous beans, the developer should include a resource reference in each component that needs access to a work manager. For more information on resource references, refer to the References topic. The component looks up a work manager using a logical name in the component, java:comp namespace, just as it looks up a data source, enterprise bean or connection factory.

The deployer binds physical work managers to logical work managers when the application is deployed.

For example, if a developer needs three thread pools to partition work between bronze, silver, and gold levels, the developer writes the component to pick a logical pool based on an attribute in the client application profile. The deployer has the flexibility to decide how to map this request for three thread pools. The deployer might decide to use a single thread pool on a small machine. In this case, the deployer binds all three resource references to the same work manager instance (that is, the same JNDI name). A larger machine might support three thread pools, so the deployer binds each resource reference to a different work manager. Work managers can be shared between multiple Java EE applications installed on the same server.

An application developer can use as many logical work managers as necessary. The deployer chooses whether to map one physical work manager or several to the logical work manager defined in the application.

All Java EE components that need to share asynchronous scope objects must use the same work manager. These scope objects have an affinity with a single work manager. An application that uses asynchronous scopes should verify that all of the components using scope objects use the same work manager.

When multiple work managers are defined, the underlying thread pools are created in a Java virtual machine (JVM) only if an application within that JVM looks up the work manager. For example, there might be ten thread pools (work managers) defined, but none are actually created until an application looks these pools up.

Important: Asynchronous beans do not support submitting work to remote JVMs.

CommonJ Work Manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a Java EE 1.4 environment, the interface does not return a new instance for each JNDI naming lookup, since this specification is not included in the Java EE specification.

Remote start of work. The CommonJ Work specification optional feature for work running remotely is not supported. Even if a unit of work implements the `java.io.Serializable` interface, the unit of work does not run remotely.

How to look up a work manager

An application can look up a work manager as follows. Here, the component contains a resource reference named `wm/myWorkManager`, which was bound to a physical work manager when the component was deployed:

```
InitialContext ic = new InitialContext();
WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
```

Inheritance Java EE contexts

Asynchronous beans can inherit the following Java EE contexts.

Internationalization context
: When this option is selected and the internationalization service is enabled, and the internationalization context that exists on the scheduling thread is available on the target thread.

Work area
: When this option is selected, the work area context for every work area partition that exists on the scheduling thread is available on the target thread.

Application profile (deprecated)
: Application profile context is not supported and not available for Java EE 1.4 applications. For Java EE 1.3 applications, when this option is selected, the application profile service is enabled, and the application profile service property, **5.x compatibility mode**, is selected. The application profile task that is associated with the scheduling thread is available on the target thread for Java EE 1.3 applications. For Java EE 1.4 applications, the application profile task is a property of its associated unit of work, rather than a thread. This option has no effect on the behavior of the task in Java EE 1.4 applications. The scheduled work that runs in a Java EE 1.4 application does not receive the application profiling task of the scheduling thread.

Security
: The asynchronous bean can be run as anonymous or as the client authenticated on the thread that created it. This behavior is useful because the asynchronous bean can do only what the caller can do. This action is more useful than a RUN_AS mechanism, for example, which prevents this kind of behavior. When you select the **Security** option, the JAAS subject that exists on the scheduling thread is available on the target thread. If not selected, the thread runs anonymously.

Component metadata
: Component metadata is relevant only when the asynchronous bean is a simple Java object. If the bean is a Java EE component, such as an enterprise bean, the component metadata is active.

The contexts that can be inherited depend on the work manager used by the application that creates the asynchronous bean. Using the administrative console, the administrator defines the sticky context policy of a work manager by selecting the services on which the work manager is to be made available.

Programming model

Work managers support the following programming models.

- **CommonJ Specification**. The Application Server Version 6.0 CommonJ programming model uses the WorkManager and TimerManager to manage threads and timers asynchronously in the Java EE 1.4 environment.
- **Asynchronous beans and CommonJ specification extensions**. The current asynchronous beans Event Source, asynchronous scopes, subsystem monitors and Java EE Context interfaces are a part of the CommonJ extension.

The following table describes the method mapping between the CommonJ and Asynchronous beans APIs. You can change the current asynchronous beans interfaces to use the CommonJ interface, while maintaining the same functions.

Table 3. Method mapping between the CommonJ and Asynchronous beans APIs. Method mapping between the CommonJ and Asynchronous beans APIs

CommonJ package	**API**	**Asynchronous beans package**	**API**
Work manager		Work manager	
Asynchronous beans	Field - IMMEDIATE (long)		Field - IMMEDIATE (int)
	Field - INDEFINITE		Field - INDEFINITE
	schedule(Work) throws WorkException, IllegalArgumentException		startWork(Work) throws WorkException, IllegalArgumentException
	schedule(Work, WorkListener) throws WorkException, IllegalArgumentException **Important:** Configure the work manager work timeout property to the value you previously specified as `timeout_ms` on `startWork`. The default timeout value is INDEFINITE.		startWork(Work, timeout_ms, WorkListener) throws WorkException, IllegalArgumentException
	waitForAll(workItems, timeout_ms)		join(workItems, JOIN_AND, timeout_ms)
	waitForAny(workItems, timeout_ms)		join(workItems, JOIN_OR, timeout_ms)
WorkItem		WorkItem	
	getResult		getResult
	getStatus		getStatus
WorkListener		WorkListener	
	workAccepted(WorkEvent)		workAccepted(WorkEvent)
	workCompleted(WorkEvent)		workCompleted(WorkEvent)
	workRejected(WorkEvent)		workRejected(WorkEvent)
	workStarted(WorkEvent)		workStarted(WorkEvent)
WorkEvent		WorkEvent	
	Field - WORK_ACCEPTED		Field - WORK_ACCEPTED

Table 3. Method mapping between the CommonJ and Asynchronous beans APIs (continued). Method mapping between the CommonJ and Asynchronous beans APIs

	Field - WORK_COMPLETED		Field - WORK_COMPLETED
	Field - WORK_REJECTED		Field - WORK_REJECTED
	Field - WORK_STARTED		Field - WORK_STARTED
	getException		getException
	getType		getType
	getWorkItem().getResult() **Important:** This API is valid only after the work is complete.		getWork
Work	(extends Runnable)	Work	(Extends Runnable)
	isDaemon		*
	release		release
RemoteWorkItem	RemoteWorkItem capability is not provided by WebSphere Application Sever. Use Distributed WorkManager in the WebSphere Extended Deployment product.	NA	
TimerManager		AlarmManager	
	resume		*
	schedule(Listener, Date)		create(Listener, context, time) ** need to convert the parameters
	schedule(Listener, Date, period)		
	schedule(Listener, delay, period)		
	scheduleAtFixedRate (Listener, Date, period)		
	scheduleAtFixedRate (Listener, delay, period)		
	stop		
	suspend		
Timer		Alarm	
	cancel		cancel
	getPeriod		
	getTimerListener		getAlarmListener
	scheduledExecutionTime		
TimerListener		AlarmListener	
	timerExpired(timer)		fired(alarm)
StopTimerListener		Not applicable	
	timerStop(timer)		
CancelTimerListener		Not applicable	
	timerCancel(timer)		

Table 3. Method mapping between the CommonJ and Asynchronous beans APIs (continued). Method mapping between the CommonJ and Asynchronous beans APIs

WorkException	(Extends Exception)	WorkException	(Extends WsException)
WorkCompletedException	(Extends WorkException)	WorkCompletedException	(Extends WorkException)
WorkRejectedException	(Extends WorkException)	WorkRejectedException	(Extends WorkException)

For more information on work manager APIs, refer to the Javadoc.

Work manager examples

Table 4. Look up work manager. Work manager

Asynchronous beans	CommonJ

`InitialContext ctx  = new InitialContext();`
`com.ibm.websphere.asynchbeans.WorkManager wm  =`
`(com.ibm.websphere.asynchbeans.WorkManager)`
`        ctx.lookup("java:comp/env/wm/MyWorkMgr");`
	`InitialContext ctx  = new InitialContext();`
	`commonj.work.WorkManager wm =`
	` (commonj.work.WorkManager)`
	`      ctx.lookup("java:comp/env/wm/MyWorkMgr");`

Table 5. Create your work using MyWork. MyWork

Asynchronous beans	CommonJ

`public class MyWork implements`
`com.ibm.websphere.asynchbeans.Work {`
`public void release() {`
`             ......`
`      }`
`    public void run() {`
`       System.out.println("Running.....");`
`   }`
	`public class MyWork implements`
	`commonj.work.Work{`
	`    public boolean isDaemon() {`
	`        return false;`
	`    }`
	`    public void release () {`
	`         .....`
	`     }`
	`   public void run () {`
	`      System.out.println("Running.....");`
	`   }`

Table 6. Submit the work. Submit work

Asynchronous beans	CommonJ

Table 6. Submit the work (continued). Submit work

MyWork work1 = new MyWork();
MyWork work2 = new MyWork();
 	
WorkItem item1;
WorkItem item2;
Item1=wm.startWork(work1);
Item2=wm.startWork(work2);
 	
// case 1: block until all items are done
ArrayList col1 = new ArrayList();
Col1.add(item1);
Col1.add(item2);
wm.join(col1, WorkManager.JOIN_AND,
WorkManager.INDEFINITE);
// when the works are done
System.out.println("work1 data="+work1.getData());
System.out.println("work2 data="+work2.getData());
 	
// you should complete case 1 before case 2
//case 2: wait up to 1000 milliseconds
for any of the items to complete.
Boolean ret = wm.join(col1,
 WorkManager.JOIN_OR, 1000);
	MyWork work1 = new MyWork();
	MyWork work2 = new MyWork();
	
	WorkItem item1;
	WorkItem item2;
	Item1=wm.schedule(work1);
	Item2=wm.schedule(work2);
	
	// case 1: block until all items are done
	Collection col1 = new ArrayList();
	col1.add(item1);
	col1.add(item2);
	wm.waitForAll(col1, WorkManager.INDEFINITE);
	// when the works are done
	System.out.println("work1 data="+work1.getData());
	System.out.println("work2 data="+work2.getData());
	
	// // you should complete case 1 before case 2
	//case 2: wait up to 1000 milliseconds
	for any of the items to complete.
	Collection finished = wm.waitForAny(col1,
	// check the workItems status
	if (finished != null) {
	  Iterator I = finished.iterator();
	  if (i.hasNext()) {
	     WorkItem wi = (WorkItem) i.next();
	      if (wi.equals(item1)) {
	        System.out.println("work1 =
	     "+ work1.getData());
	      } else if (wi.equals(item2)) {
	        System.out.println("work1 =
	     "+ work1.getData());
	      }
	  }
	}

Table 7. Create a timer manager. Timer manager

Asynchronous beans	**CommonJ**

InitialContext ctx = new InitialContext();
com.ibm.websphere.asynchbeans.WorkManager wm =
    (com.ibm.websphere.asynchbeans.WorkManager)
      ctx.lookup("java:comp/env/wm/MyWorkMgr");
 	
AsynchScope ascope;
Try {
  Ascope = wm.createAsynchScope("ABScope");
} Catch (DuplicateKeyException ex)
{
   Ascope = wm.findAsynchScope("ABScope");
   ex.printStackTrace();
}
 	
// get an AlarmManager
AlarmManager aMgr= ascope.getAlarmManager();
	InitialContext ctx = new InitialContext();
	Commonj.timers.TimerManager tm =
	(commonj.timers.TimerManager)
	ctx.lookup("java:comp/env/tm/MyTimerManager");

Table 8. Fire the timer. Fire timer

Asynchronous beans	**CommonJ**

Table 8. Fire the timer (continued). Fire timer

// create alarm
ABAlarmListener listener = new ABAlarmListener();
Alarm am =
  aMgr.create(listener, "SomeContext", 1000*60);
	// create Timer
	TimerListener listener =
	new StockQuoteTimerListener("qqq",
	"johndoe@example.com");
	Timer timer = tm.schedule(listener, 1000*60);
	
	// Fixed-delay: schedule timer to expire in
	// 60 seconds from now and repeat every
	// hour thereafter.
	Timer timer = tm.schedule(listener, 1000*60,
	1000*30);
	
	// Fixed-rate: schedule timer to expire in
	// 60 seconds from now and repeat every
	// hour thereafter
	Timer timer = tm.scheduleAtFixedRate(listener,
	1000*60, 1000*30);

Timer managers

The timer manager combines the functions of the asynchronous beans alarm manager and asynchronous scope. So, when a timer manager is created, it internally uses an asynchronous scope to provide the timer manager life cycle functions.

You can look up the timer manager in the Java Naming and Directory Interface (JNDI) name space. This capability is different from the alarm manager that is retrieved through the asynchronous beans scope. Each lookup of the timer manager returns a new logical timer manager that can be destroyed independently of all other timer managers.

A timer manager can be configured with a number of thread pools through the administrative console. For deployment you can bind this timer manager to a resource reference at assembly time, so the resource reference can be used by the application to look up the timer manager.

The Java code to look up the timer manager is:

```
InitialContext ic = new InitialContext();
TimerManager tm = (TimerManager)ic.lookup("java:comp/env/tm/TimerManager");
```

The programming model for setting up the alarm listener and the timer listener is different. The following code example shows that difference.

Table 9. Set up the timer listener. Programming model for setting up the timer listener

Asynchronous beans	CommonJ

Table 9. Set up the timer listener (continued). Programming model for setting up the timer listener

public class ABAlarmListener implements
AlarmListener {
public void fired(Alarm alarm) {
System.out.println("Alarm fired.
Context =" + alarm.getContext());
}
	public class StockQuoteTimerListener implements
	TimerListener {
	String context;
	String url;
	public StockQuoteTimerListener(String context,
	String url){
	this.context = context;
	This.url = url;
	}
	public void timerExpired(Timer timer) {
	System.out.println("Timer fired. Context ="+
	((StockQuoteTimerListener)timer.getTimerListener())
	.getContext());
	
	}
	public String getContext() {
	return context;
	}
	}

Example: Using connections with asynchronous beans

An asynchronous bean method can use the connections that its creating Java Platform, Enterprise Edition (Java EE) component obtained using java:comp resource references.

For more information on resource references, refer to the References topic. The following is an example of an asynchronous bean that uses connections correctly:

```
class GoodAsynchBean
{
 DataSource ds;
 public GoodAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource
  // as class instance data.
  InitialContext ic = new InitialContext();
  // it is assumed that the created Java EE component has this
  // resource reference defined in its deployment descriptor.
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
 }
 // When the asynchronous bean method is called, get a connection,
 //  use it, then close it.
 void anEventListener()
 {
  Connection c = null;
  try
  {
   c = ds.getConnection();
   // use the connection now...
  }
  finally
  {
   if(c != null) c.close();
  }
 }
}
```

The following example of an asynchronous bean that uses connections incorrectly:

```
class BadAsynchBean
{
 DataSource ds;
 // Do not do this. You cannot cache connections across asynch method calls.
 Connection c;

 public BadAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource as
  // class instance data.
  InitialContext ic = new InitialContext();
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
  // here, you broke the rules...
  c = ds.getConnection();
 }
 // Now when the asynch method is called, illegally use the cached connection
 // and you likely see J2C related exceptions at run time.
 // close it.
 void someAsynchMethod()
 {
  // use the connection now...
 }
}
```

Chapter 5. Bean Validation

The Bean Validation API is introduced with the Java Enterprise Edition 6 platform as a standard mechanism to validate Enterprise JavaBeans in all layers of an application, including, presentation, business and data access. Before the Bean Validation specification, the JavaBeans were validated in each layer. To prevent the reimplementation of validations at each layer, developers bundled validations directly into their classes or copied validation code, which was often cluttered. Having one implementation that is common to all layers of the application simplifies the developers work and saves time.

Bean Validation

The Bean Validation API is introduced with the Java Enterprise Edition 6 platform as a standard mechanism to validate JavaBeans in all layers of an application, including presentation, business, and data access.

Before the Bean Validation specification, JavaBeans were validated in each layer. To prevent the re-implementation of validations at each layer, developers bundled validations directly into their classes or copied validation code, which was often cluttered. Having one implementation that is common to all layers of the application simplifies the developers work and saves time.

The Bean Validation specification defines a metadata model and an API that are used to validate JavaBeans for data integrity. The metadata source is the constraint annotations defined that can be overridden and extended using XML validation descriptors. The set of APIs provides an ease of use programming model allowing any application layer to use the same set of validation constraints. Validation constraints are used to check the value of annotated fields, methods, and types to ensure that they adhere to the defined constraint.

Constraints can be built in or user-defined. Several built-in annotations are available in the javax.validation.constraints package. They are used to define regular constraint definitions and for composing constraints. For a list of built-in constraints, see the topic, Bean validation built-in constraints. For more details about the Bean Validation metadata model and APIs see the JSR 303 Bean Validation specification document.

The following example is a simple Enterprise JavaBeans (EJB) class that is decorated with built-in constraint annotations.

```
public class Home  {
   @Size(Max=20)
    String builder;
    @NotNull @Size(Max=20)
    String address;

    public String getAddress() {
         return address;
    }

    public String getBuilder() {
         return address;
    }
    public String setAddress(String newAddress) {
         return address = newAddress;
    }

    public String setBuilder(String newBuilder) {
         return builder = newBuilder;
    }
}
```

The @Size annotations on builder and address specify that the string value assigned should not be greater 20 characters. The @NotNull annotation on address indicates that it cannot be null. When the Home object is validated, the builder and address values are passed to the validator class defined for the @Size annotation. The address value is also be passed to the @NotNull validator class. The validator classes handle checking the values for the proper constraints and if any constraint fails validation, a ConstraintViolation object is created, and is returned in a set to the caller validating the Home object.

Validation APIs

The javax.validation package contains the bean validation APIs that describe how to programmatically validate JavaBeans.

ConstraintViolation is the class describing a single constraint failure. A set of ConstraintViolation classes is returned for an object validation. The constraint violation also exposes a human readable message describing the violation.

ValidationException are raised if a failure happens during validation.

The Validator interface is the main validation API and a Validator instance is the object that is able to validate the values of the Java object fields, methods, and types. The bootstrapping API is the mechanism used to get access to a ValidatorFactory that is used to create a Validator instance. For applications deployed on the product, bootstrapping is done automatically. There are two ways for applications to get the validator or the ValidatorFactory. One way is injection, for example, using the @Resource annotation, and the other way is the java: lookup.

The following example uses injection to obtain a ValidatorFactory and a Validator:

```
@Resource ValidatorFactory _validatorFactory;
@Resource Validator _validator;
```

Attention: When using @Resource to obtain a Validator or ValidatorFactory, the authenticationType and shareable elements must not be specified.

The following example uses JNDI to obtain a ValidatorFactory and a Validator:

```
ValidatorFactory validatorFactory = (ValidatorFactory)context.lookup("java:comp/ValidatorFactory");
Validator validator = (Validator)context.lookup("java:comp/Validator");
```

Constraint metadata request APIs

The metadata APIs support tool providers, provides integration with other frameworks, libraries, and Java Platform, Enterprise Edition technologies. The metadata repository of object constraints is accessed through the Validator instance of a given class.

XML deployment descriptors

Besides declaring constraints in annotations, support exists for using XML to declare your constraints.

The validation XML description is composed of two kinds of xml files. The META-INF/validation.xml file describes the bean validation configuration for the module. The other XML file type describes constraints declarations and closely matches the annotations declaration method. By default, all constraint declarations expressed through annotations are ignored for classes described in XML. It is possible to force validation to use both the annotations and the XML constraint declarations by using the ignore-annotation="false" setting on the bean. The product ensures that application modules deployed containing a validation.xml file and constraints defined in XML files are isolated from other module validation.xml and constraint files by creating validator instances specific to the module containing the XML descriptors.

Advanced bean validation concepts

The Bean Validation API provides a set of built-in constraints and an interface that enables you to declare custom constraints. This is accomplished by creating constraint annotations and declaring an annotation on a bean type, field, or property. Composing constraints is also done by declaring the constant on another constraint definition.

The following example shows creating a CommentChecker constraint that is defined to ensure a comment string field is not null. The comment text is enclosed by brackets, such as [*text*].

```
package com.my.company;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import javax.validation.Constraint;
import javax.validation.Payload;

@Documented
@Constraint(validatedBy = CommentValidator.class)
@Target({ METHOD, FIELD })
@Retention(RUNTIME)
public @interface CommentChecker {
       String message() default "The comment is not valid.";
       Class<?>[] groups() default {};
       Class<? extends Payload>[] payload() default {};
      ...}
```

The next example shows the constraint validator that handles validating elements with the @CommentChecker annotation. The constraint validator implements the ConstraintValidator interface provided by the Bean Validation API.

```
package com.my.company;
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
public class CommentValidator implements ConstraintValidator<CommentChecker, String> {
    public void initialize(CommentChecker arg0) {
    }
    public boolean isValid(String comment, ConstraintValidatorContext context) {
        if (comment == null) {
            // Null comment is not allowed, fail the constraint.
            return false;
        }
        if (!comment.contains("[") && !comment.contains("]")) {
            // Can't find any open or close brackets, fail the constraint
            return false;
        }
        // Ignore leading and trailing spaces
        String trimmedComment = comment.trim();
        return // validate '[' prefix condition
               trimmedComment.charAt(0) == '[' &&
               // validate ']' suffix condition
               trimmedComment.charAt(trimmedComment.length()!-1) == ']';
    }
}
```

After the @CommentChecker is defined, it can be used to ensure that the comment string field is a valid comment based on the CommentValidator isValid() implementation. The following example shows the use of the @CommentChecker constraint. When the myChecker bean is validated, the comment string is validated by the CommentValidator class to ensure the constraints defined are met.

```
package com.my.company;
public myChecker {

    @CommentChecker
    String comment = null;
    ...
}
```

The product provides a specific bean validation provider, but it might be necessary for an application to use or require another provider.

This method can be accomplished by using the validator methods to set the provider programmatically and create a validation factory. Or, by using the validation.xml default-provider element. The specific provider that is defined and used to create the validation factory and not the default provider provided by the application server in the default implementation. If you want to ensure that the user-provided implementation does not conflict with the default implementation, the server or application class loading parameter, the class loader order should be set to be loaded with local class loader first (parent last). See additional information in the class loading documentation on how to set this setting.

The Bean Validation specification indicates that if more than one validation.xml file is found in the class path, a ValidationException occurs. However, WebSphere Application Server supports an environment where multiple teams develop modules that are assembled and deployed into the Application Server together. In this environment, all EJB modules within an application are loaded with the same class loader and it is possible to configure the application class loaders so that all EJB and web archive (WAR) modules are loaded by a single class loader. Because of this, the product provides support for multiple validation.xml files in the same class path.

When an application using bean validation and XML descriptors contains multiple EJB modules and web modules, each validation.xml file is associated with a validation factory that is specific to that module. In this environment, any constraint-mapping elements that are defined are only looked up in the module where the validation.xml file is defined. For example, if an EJB module building.jar contains a META-INF/validation.xml file and the validation.xml file defined the following constraints, both the META-INF/constraints-house.xml and META-INF/constraints-rooms.xml files must also be located in the building.jar file:

```
<constraint-mapping>META-INF/constraints-house.xml</constaint-mapping>
<constraint-mapping>META-INF/constraints-rooms.xml</constraint-mapping>
```

The exception to this behavior is when all bean validation constraints classes and configuration are visible to all application modules. In a case where a single validation.xml file is defined in an EAR file, and no other validation.xml files are visible in a module's class path, any module that creates a validator factory or validator will use the validation.xml file that is defined in the EAR file. This makes it possible for other modules to create a validator factory that uses the validation.xml file of another module, if the class path has been configured so that both modules are visible on the same class path and only one validation.xml file is visible to those modules.

For a more detailed understanding about the Bean Validation APIs and metadata see the JSR 303 Bean Validation specification document.

Chapter 6. Communications Enabled Applications

Communications Enabled Applications (CEA) is a functionality that provides the ability to add dynamic web communications to any application or business process. The product provides a suite of integrated telephony and collaborative web services that extends the interactivity of enterprise and web commerce applications. With the CEA capability, enterprise solution architects and developers can use a single core application to enable multiple modes of communication. Enterprise developers do not need to have extensive knowledge of telephony or Session Initiation Protocol (SIP) to implement CEA. The CEA capability delivers call control, notifications, and interactivity and provides the platform for more complex communications.

Communications Enabled Applications concepts

Communications Enabled Applications (CEA) is a programming model that provides the ability to add dynamic web communications to any application or business process.

You can take advantage of integrated telephony and collaborative web services to extend the interactivity of Enterprise and web commerce applications. With the CEA capability, Enterprise solution architects and developers can use a single core application to enable multiple modes of communication. Enterprise developers do not need to have extensive knowledge of telephony or Session Initiation Protocol (SIP) to implement CEA. The CEA capability delivers call control, notifications, and interactivity and provides the platform for more complex communications.

Using this simplified programming model for adding web-based communications, Enterprise developers can do the following:

- Enable any application to quickly add communications support; for example, click-to-call integration
- Enable shared sessions between end users and the company
- Push relevant session data for application use; for example, customer phone numbers
- Deliver automated notifications and instant messaging support
- Provide enterprise-grade security, scalability, and high availability
- Integrate with customer private branch exchange (PBX) systems

CEA has two main services, telephony access and multimodal web interaction:

- Telephony access allows you to create a unified communications environment from within business applications to increase the efficiency of processes, reduce communications errors, and optimize business interactions in real time. CEA provides telephony access through a REST interface, through a web services client, and by using click-to-call widgets.
- Multimodal web interaction allows you to provide session linking (shared sessions) between users browsing the same website from different locations. With session linking, users can interact dynamically in collaborative ways, such as cobrowsing or coshopping web sessions. Commerce web sites can use this service to provide product or customer support, while protecting information on the internal site. Commerce sites can use the collaborative shopping experience to attract more customers to their sites. With a combination of click-to-call functionality and multimodal interaction, you can support two-way synchronized text forms between the user and a customer service representative (CSR).

Note: CEA Dojo widgets encapsulate various user interfaces that are based on the CEA Rest API. These widgets are deprecated. You can also obtain the source for these widgets from developerWorks.

CEA is based on SIP-enabled services that use Representation State Transfer (REST) servlets and web services in a converged HTTP and SIP application. CEA includes a library of Dojo-style widgets for use in web applications. Dojo widgets are prepackaged components of JavaScript and HTML code that add interactive features that work across platforms and browsers. CEA widgets are extensible, allowing developers to customize them to handle more advanced tasks.

The CEA samples package includes three different sample applications that you can use to explore the telephony access and multi-modal web interaction services. To learn more, see the information on accessing the samples and setting up the communications enabled application samples.

CEA call flow

In a Web telephony session, users can make phone calls using the ClicktoCall widget in their Web browsers. The call flow diagram illustrates the Communications Enabled Applications (CEA) call flow.

Note: The system application must first be configured with the IP private branch exchange (PBX) address.

The following configuration applies to this call flow:

1. A user clicks the ClickToCall widget, which sends an HTTP REST request.
2. The Web container calls the system application.
   - An HTTP servlet interprets the REST request.
3. The system application sends SIP messages to the IP PBX using the ECMA TR/87 standard protocol.

   In addition to TR/87, PBX vendors can choose to expose a web service interface based on a Web Services Description Language (WSDL) file that is provided with CEA. This WSDL file is included in the installation path, *app_server_root*/systemApps/commsvc.ear/commsvc.rest.war/WEB-INF/wsdl/ControllerService.wsdl. Using this WSDL file enables you to configure CEA to call out to a PBX through a web service instead of relying on TR/87.

   Restriction: The IP PBX must support the ECMA TR/87 protocol or the CEA web service interface.
4. The IP PBX notifies the user agent client (UAC) to call the user agent server (UAS).
5. A call is established between the two users.

   Note: UAC and UAS are SIP instances.

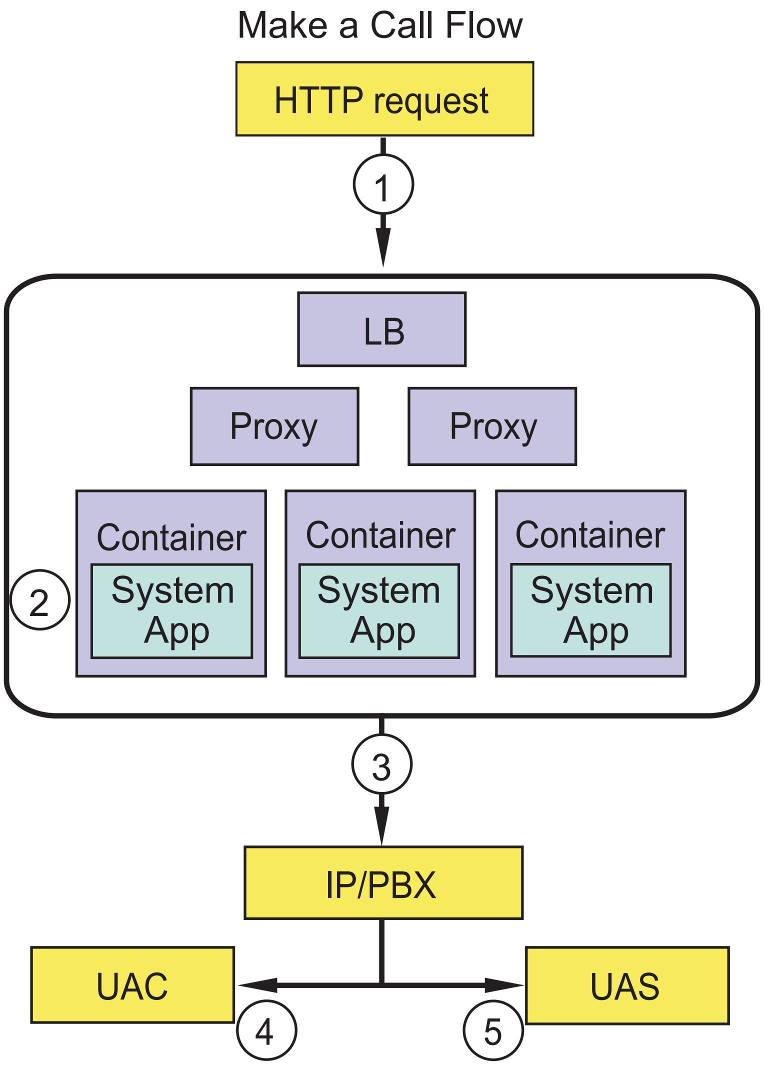


Figure 5. CEA call flow

CEA collaboration flow

A Communications Enabled Applications (CEA) Web collaboration session allows dynamic interaction between users sharing linked browser sessions. Users connect their Web sessions using the CEA collaboration widget. Use the collaboration flow illustrated in this topic to understand how live Web collaboration works.

1. User A initiates collaboration by clicking the collaboration widget, which sends a specific HTTP REST request.
2. The container calls the system application:
   - User A is placed in the user registry.
   - User A's session is established.
3. The response to User A includes a uniform resource locator (URI) for peers to start collaboration.
4. User A sends User B the “for peer collaboration URI”.
   - The “for peer collaboration URI” contains a *nonce*, a unique identifier which is particular to the collaboration session. The nonce helps ensure security in the collaboration session.
5. User B responds by sending a request with that URI.

6. The container calls the system application:
   - User B is placed in the user registry.
   - User A is found in the user registry.
   - A “link” is established between them.
7. A response is sent to User B.
   - Includes URI to exchange data.
   - Activates modal windows in each widget.
8. User B highlights text, scrolls, or fills in a form.
9. User B's widget sends these events through the send URI.
10. The container sends data to User A's session.
11. User A's widget polls for events with the fetch URI.
12. User B's events are captured in User A's widget.

The following diagram shows the collaboration flow between linked user sessions.

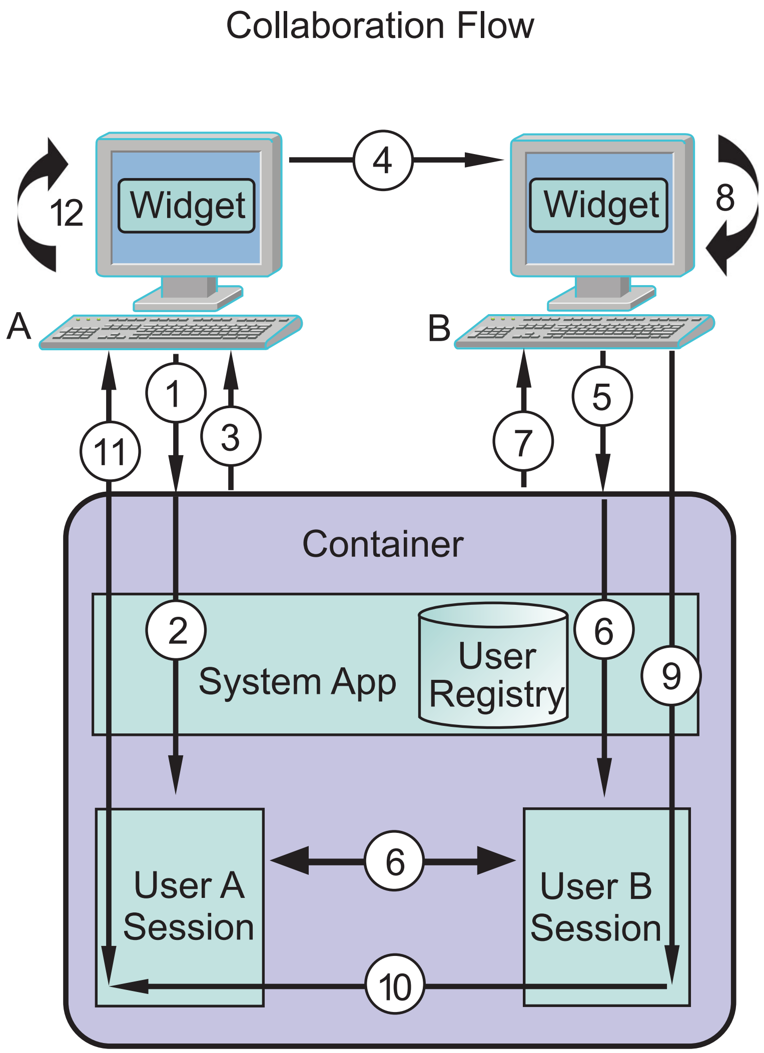


Figure 6. CEA Web collaboration

CEA iWidgets

An iWidget is a browser-oriented component designed to work within the framework defined by the iWidget specification. Such a component only occupies a portion of the overall working canvas and is typically designed in a way that makes it easy for the canvas assembler to connect the iWidget to other iWidgets on the canvas.

The ClickToCall, CallNotification and Cobrowse widgets have each been wrapped according to the iWidget specification and packaged together in an iWidget package (WAR file). This package is included in the installation path, *app_server_root*/installableApps/cea.war.

Attention: For instructions on how to publish this iWidget package, see the documentation for the iWidget container that you plan to use.

After you publish the iWidget package and place the iWidget on a page, the widget loads to the following state:

```
The service is currently unavailable.
```

For most scenarios the iWidget container runs separately from the Communications Enabled Applications (CEA) application server. This type of environment requires the use of a proxy to proxy the request from the widget to the Representational State Transfer (REST) service. The proxy mapping must be configured to proxy GET, POST, PUT, and DELETE methods to:

```
cea_server:cea_server_port:/commsvc.rest/CommServlet/*
```

After the proxy is configured, go to the edit settings, view and then choose to edit the widget settings.

From the edit settings page you can configure the various widget attributes to be used by the widget on the current page. From this page modify the ceaContextRoot setting to point the proxy URL for the REST service. After you specify the correct ceaContextRoot value, save the settings for the widget to update to the default state.

For some scenarios it might make sense to hard code the widget attributes before publishing the iWidget package. To do so, open the cea.war file and modify itemSet "userPrefs" in clickToCall.xml, callNotification.xml, or cobrowse.xml. Save the changes, and publish the cea.war file to the iWidget container.

```
<iw:itemSet id="userPrefs">
      <iw:item id="ceaContextRoot" value="/commsvc.rest"/>
      <iw:item id="widgetNumber" value=""/>
      <iw:item id="enableCollaboration" value="false"/>
      <iw:item id="defaultCollaborationUri" value=""/>
      <iw:item id="canControlCollaboration" value="false"/>
      <iw:item id="highlightElementList" value="DIV,SPAN,TR,TH,TD,P"/>
      <iw:item id="isHighlightableCallback" value=""/>
      <iw:item id="isClickableCallback" value=""/>
</iw:itemSet>
```

For more information about iWidgets, see the IBM Mashup Center wiki.

Collaboration Dialog

The Collaboration Dialog allows two users to share information over linked browser sessions. Use this topic to understand how the Collaboration Dialog works.

The Collaboration Dialog widget is used by the ClickToCall, CallNotification, and Cobrowse widgets to provide peer-to-peer page sharing and allow one user to control a collaboration session. With the Collaboration Dialog, page sharing is not screen sharing because each browser makes its own connection to the server for the content.

The Collaboration Dialog interface has three parts. The widget toolbar shows the browser and collaboration controls for driving the peer-to-peer session. The content pane is the area that loads the pages that can be shared with the peer. The status bar displays information for the collaboration session.

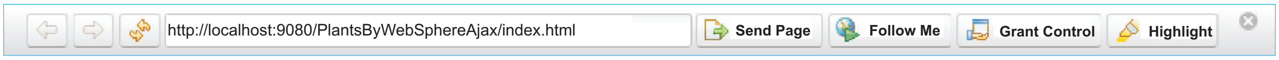
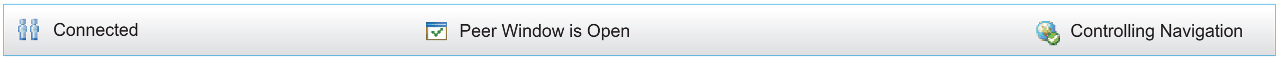

http://localhost:9080/PlantsByWebSphereAjax/index.html

Figure 7. Collaboration Dialog toolbar


Connected

Peer Window is Open

Controlling Navigation

Figure 8. Collaboration Dialog status bar

When two users are involved in a collaboration session, the initiator of the session has control and is the active peer. The passive peer can see the actions taken by the active peer. If the **Grant Control** option (see Collaboration controls) is used, then the active and passive roles can switch. Only one user can have control at a time.

Browser controls

The Collaboration Dialog give users the following controls to enable page navigation:

- Click the **Back** icon to go back one page. If **Follow Me** is enabled, the peer's window is also updated to load the same page.
- Click the **Forward** icon to go forward one page. If **Follow Me** is enabled, the peer's window is also updated to load the same page.
- Click the **Refresh** icon to reload the current page. If **Follow Me** is enabled, the peer's window is also updated to reload the same page.
- Type a URL into the Location bar and press enter to browse to the page. This URL must be on the same domain as the page used to display the Collaboration Dialog.

Collaboration controls

The Collaboration Dialog toolbar contains the following collaboration controls: **Send Page, Follow Me, Grant Control**, and **Highlight**.

- The **Send Page** icon allows the active peer, the user driving the collaboration session, to send the URL of the current page to the passive peer. The send page function is useful when the active peer only needs to show the peer certain pages and not the path taken to browse to that page. To send the page, click the **Send Page** icon and the current URL of the page is sent to the passive peer.

  Remember: The following behavior applies to the send page function:

  - The URL of the current page is sent and not the current text in the Location bar.
  - Any customization on the current page, for example, a highlight or any changes to the page made from Ajax requests, are reset when the page reloads.
- The **Follow Me** icon allows the active peer to send clicks to the passive peer. The Collaboration Dialog widget captures any clicks that the active peer makes on the page when Follow Me is enabled. This information is then sent to the passive peer to simulate the clicks in their Collaboration Dialog. To stop sending this information, the active peer toggles the **Follow Me** control. The follow me function is useful when:
  - The active peer needs to show the passive peer how to navigate to a page.
  - Navigating sites that are built using Web 2.0 technologies; for example, sites using Ajax-style requests to update content instead of full page refreshes.

- The **Grant Control** icon allows the active peer to transfer control to the passive peer, providing the passive peer has the appropriate permission to drive the collaboration session. Once the active and passive roles are switched, the new active peer will have their collaboration controls enabled. If the passive peer does not have permission to drive the collaboration session, the **Grant Control** icon remains disabled.
- The **Highlight** icon allows the active peer to highlight a section of the page currently displayed by the Collaboration Dialog. This information is then sent to the peer, and the same section is highlighted on their page. Both the active peer and the passive peer can perform highlights. To perform a highlight, click the **Highlight** icon, and then move the cursor over the section of the page to be highlighted. As the cursor moves, the sections change color to show what can be highlighted. Once the user is at the section they want to highlight, left-click to send the highlight to the peer.

Collaboration Dialog status

The Collaboration Dialog has the following status indicators in the status bar:

- The **Connection** status icon displays the connection status of the Collaboration Dialog. When either peer disconnects, this status is updated to show the disconnected status. States included are: Connected and Disconnected.
- The **Peer Window** status icon updates as either peer opens and closes their Collaboration Dialog. States included are: Peer window is open and Peer window is closed.
- The **Collaboration Action** status icon displays collaboration action and whether the user is currently controlling the session. States included are: Controlling navigation, Cobrowsing web, Follow me, and Coauthoring form.

gotcha:

- When URLs are passed between the Collaboration Dialog peers using the Send Page function, the entire URL is sent to the peer. If the browser for the active peer is on the same machine as the server, and the active peer uses local host to access the server, this environment causes issues for the passive peer if they are on separate machines. Because the entire URL is passed, the passive peer attempts to access the page using local host and a failure occurs.

  When you test the Collaboration Dialog, and the browsers are on separate machines, you must specify the host name or IP address of the server that is accessible to both peers in the URL instead of the local host. This process is necessary under the following conditions:

  – When you access a page that contains an embedded widget

  – When you enter addresses into the Location Bar of the Collaboration Dialog

  Note: The sendPageUrlRewriteCallback attribute is added to all the widgets to enable the application to provide the name of a callback JavaScript function:

  sendPageUrlRewriteCallback
  A string containing the name of the callback function to run when send page is called to rewrite the current URL.

  This is useful when the peers are accessing the application on different domains or through a proxy.
- If you want to browse pages by way of Hypertext Transfer Protocol Secure (HTTPS) using the Collaboration Dialog, the page containing the embedded CEA widget must also be accessed by means of HTTPS. If you launch one of the CEA widgets from a page accessed through HTTP, and then during the Collaboration Dialog session you view a page through HTTPS, the Follow Me and Highlight functionality will not work for that page due to the JavaScript same origin policy. This same issue occurs if you load the page containing the embedded CEA widget through HTTPS, and then you try to view a page in the Collaboration Dialog through HTTP.

Collaborative two-way forms

You can use attributes to customize web-based two-way forms.

The Communication Enabled Applications (CEA) two-way form widget is used to create HTML forms in which two people can collaboratively input text and validate entries. Both users can see the same form. The fields in the form change in response to input provided by either person.

The *writer* is the user who is responsible for driving the interaction between the two users.

The *reader* is the user who is responsible for providing information to the writer. The reader can provide information verbally to the writer, who copies the information into the form's fields. Since updates to the fields are visible to the reader, the reader can confirm or validate the correctness of the information. The reader can be prompted to enter sensitive information into the form, such as credit card numbers. Such private information is generally filtered so that the writer cannot see it.

Two-way form widgets

Any Dijit form widgets and their subclass widgets that are part of a two-way form automatically support two-way editing. A widget supporting two-way editing must have an ID specified. You can specify additional attributes on the widgets to expand their capabilities.

ceaCollabWriteAccess

In a two-way form, fields might exist that only one user should have write access to. For example, a two-way interaction might involve a salesperson who is responsible for submitting the form, designated as the writer, and a customer who is responsible for filling out certain portions of the form, designated as the reader. The reader might be prompted to enter credit card information, for example, in a particular field. This field must not be editable by the writer. To ensure this, specify the input field setting for the ceaCollabWriteAccess attribute to "reader":

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox" ceaCollabWriteAccess="reader" />
```

There might also be input fields that the writer fills out that the reader must not have access to. In this case, specify the input field setting for the ceaCollabWriteAccess attribute to "writer":

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox" ceaCollabWriteAccess="writer" />
```

Important: Use this attribute in conjunction with the ceaCollabValidation attribute to ensure that only one user can change a particular field; thereby preventing both users from being able to validate the same field.

ceaCollabFilter

The ceaCollabFilter attribute is used to specify a JavaScript method that is used to mask values. This is useful for fields that contain sensitive information that only one user should be allowed to see (for example, Social Security numbers). If the attribute has the value `default`, a default masking function is used that replaces every character of input with an asterisk. Otherwise, the value of the attribute is used to call a method that takes a string (the value of the input field) and is expected to return a masked version of that value.

For example, consider the following JavaScript method, used here as a masking method:

```
function mask(value) {
 return "XXXX";
}
```

You can specify that this masking function be used with a text input field by specifying it in the ceaCollabFilter attribute:

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox" ceaCollabFilter="mask" />
```

ceaCollabValidation

Two-way form functionality allows for validation to occur on any input field. This means that any change to the input field by one user will require another user to accept or decline the changes. You can submit the form only after all input fields that require validation have been accepted.

By default, the CEA TwoWayForm widget provides for a simple validation widget that appears alongside an input field widget when validation is required. To enable this widget, simply set the value of the ceaCollabValidation attribute on the input field widget to `default`, for example:

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox" ceaCollabValidation="default" />
```

In some cases, you might not want to use the default validation widget. You can create your own by subclassing `cea.widget.validation.ValidationWidget` and overriding methods related to creating, showing, and hiding the validation widget. For more information about how a custom class is implemented, see the JavaScript comments in *app_server_root*`\etc\cea\javascript\ceadojo\cea\widget\validation\ValidationWidget.js`.

ceaCollabName

When validation is defined for an input field, an alert notifies the writer when one or more input fields have not been validated by the reader. By default, this alert only lists the input field IDs. If a more descriptive label is needed, use the ceaCollabName attribute on the input field widget, for example:

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox" ceaCollabValidation="default" />
```

ceaCollabHandleRemoveNotification/ceaCollabHandleShowNotification

When any field in a two-way form changes, the user gets a notification. By default, the field is highlighted, and the appearance of the highlighting is determined by the two-way form CSS (the `TwoWayForm.css` file referred to earlier in this topic). However, you can change the highlight styling in the following two ways:

- Changing the styling in `TwoWayForm.css` or, creating a new CSS file that redefines the styles present in `TwoWayForm.css`
- Defining alternative notification methods

Defining alternative notification methods gives you more control over how notifications are presented to the user, if at all. Define two methods, both of which accept a Dijit widget object. The first method is used to control removing the notification, and the second is used to control showing the notification. For example, consider the following JavaScript methods:

```
function removeFunc(widget) {
 ceadijit.hideTooltip (widget.domNode);
}
function showFunc(widget) {
 ceadijit.showTooltip ("Value changed", widget.domNode);
}
```

The alternative notification methods handle showing and hiding a tooltip instead of the default notification method that involves highlighting a field. To use these methods, specify them in the ceaCollabHandleRemoveNotification and ceaCollabHandleShowNotification attributes on the desired input field:

```
<input type="text" name="textName" id="textName" value="" size="30"
 ceadojoType="dijit.form.TextBox"
 ceaCollabHandleRemoveNotification="removeFunc"
 ceaCollabHandleShowNotification="showFunc" />
```

REST APIs in CEA

You can use the Communications Enabled Applications (CEA) feature to integrate either web telephony or web collaboration into applications using a Representational State Transfer (REST) API. REST is a network architecture that defines how resources on the Internet are accessed. REST is not a standard, but uses common Internet standards such as HTTP and HTML.

REST API overview

The following tables show the REST APIs and commands used in CEA, including a generic list of each aspect of the requests and responses. Also see the information on accessing telephony and data sharing browser sessions using REST APIs.

Table 10. REST API overview.

This table shows an overview of the REST API.

HTTP Method and URI	Description
PUT /collaborationSession	Enable collaboration
GET /collaborationSession; <encodedSession>	Get collaboration status
GET /collaborationSession; <encodedPeerAddressOfRecord>	Start a collaboration session with a peer
DELETE /collaborationSession; <encodedSession>	End a collaboration session
POST /collaborationSession/data; <encodedSession>	Send data to the collaboration peer
GET /event; <encodedSession>	Retrieve event data (call status, collaboration status, collaboration data)
PUT /call	Make a call
GET /call; <encodedSession>	Get status on an active call
DELETE /call; <encodedSession>	End a call
PUT /callNotification	Register for call notification
GET /callNotification; <encodedSession>	Get call notification information
DELETE /callNotification; <encodedSession>	Unregister for call notification

URI parameters

URI parameters are used by the REST API. These parameters are located after the path portion of the request URI; for example:

```
http://host:port/commsvc.rest/CommServlet/collaborationSession?JSON=true&addressOfRecord=tel:987654321
```

Table 11. URI parameters.

This table shows URI parameters and their descriptions.

Parameter	Applicable HTTP Method and URI	Description
JSON	All	Optional parameter that you can use on any REST request. When the JSON parameter is detected and set to true, the REST response is formatted as JSON; otherwise XML. The presence of this parameter does not affect the format of the request.
addressOfRecord	PUT /collaborationSession	Optional parameter when enabling collaboration. It allows the specified value to be used as an identifier of the session. This identifier gets exposed to other users of collaboration. If absent, a random identifier is created. Whether specified or not, the identifier is returned in the response as the collaborationId.

Table 11. URI parameters (continued).

This table shows URI parameters and their descriptions.

Parameter	Applicable HTTP Method and URI	Description
peerAddressOfRecord	GET /collaborationSession	Required parameter when starting a collaboration session with a peer. It represents the identifier of the peer to which a collaboration must be established. This identifier is the collaborationId that was returned to the peer when the peer enabled collaboration.
returnAllEvents	GET /event; <encodedSession>	Optional parameter used to get new events. If set to true, all available events are returned. Otherwise, only the next single event is returned. If this parameter is not used, the default is to return a single event.
ceaVersion	All	Required parameter used to declare the version of the client sending the request. It is used to support backward compatibility.
replaceRequest	GET /event; <encodedSession>	Optional parameter used to force a new GET /event request to replace an outstanding GET /event request. The outstanding request gets a 304 error response.

Request fields

With some REST APIs, additional information is provided in the body of the HTTP message, which must be formatted as either XML or JSON. The following table explains those request fields. Some fields are specific to web telephony or web collaboration while others apply to both. As another reference, see the schema that describes the REST requests and response format. It is called CommServletSchema.xsd and can be found in the WebSphere Application Server installation directory under `/etc/cea/schema`.

Table 12. Request fields.

This table shows request fields and their descriptions.

Request fields	Applicable HTTP Method and URI	Description
addressOfRecord	PUT /call PUT /callNotification	Required when making a call or registering for call notification. It represents the address of record of the phone making the call, or the phone registering for call notification. Common to all address of record references in this REST API, the values can be either SIP or TEL URIs.
peerAddressOfRecord	PUT /call	This field is required when making a call. It represents the address of record of the phone to be called.
collaborationData	POST /event; <encodedSession>	This field is required when sending data to a collaboration peer. It represents the data being sent.

Table 12. Request fields (continued).

This table shows request fields and their descriptions.

Request fields	Applicable HTTP Method and URI	Description
enableCollaboration	PUT /call PUT /callNotification	Optional when either making a call or registering for call notification. It allows collaboration to be enabled at the same time.
peerDeviceControlled	PUT /call	By default, PUT /call causes the device associated with the addressOfRecord to be controlled and originate the call. When peerDeviceControlled is set to true, the device associated with the peerAddressOfRecord is controlled and originates the call.

Response fields

The response to the REST API is described in the following table. As with the request fields, some response fields are specific to web telephony or web collaboration, while others apply to both. Many of the response fields include URIs that can be used in follow on REST requests. They are encoded to ensure that subsequent requests, related to the originally passed in address of record, remain associated with the same session. As another reference, see the schema that describes the REST requests and response format. It is called CommServletSchema.xsd and can be found in the WebSphere Application Server installation directory under `/etc/cea/schema`.

Table 13. Response fields.

This table shows response fields and their descriptions.

Response field	Applicable HTTP Method and URI	Description
infoMsg	All	Message indicating the results of the requested operation.
returnCode	All	Number that represents the result of the requested operation. For example, when a request operation completes successfully, the response is 200. This full list of return codes can be seen in the REST interface schema. See table 5 for a complete list.

Table 13. Response fields (continued).

This table shows response fields and their descriptions.

Response field	Applicable HTTP Method and URI	Description
eventList	GET /event	Array of events (web collaboration data available, web collaboration status or call status changes) used in response to GET /event. Each event has three fields: type, data, and infoMsg. The different types include call status, collaboration status and collaboration data. The data is dependent on the event type, but is similar to the REST response fields of callStatus and collaborationStatus, or the data sent with POST /event. The infoMsg is additional information about the event. The following is a JSON formatted eventList: `[{"type":2, "data": "information from peer", "infoMsg": "Successfully fetched data"}]` Possible event types based on their enum values include: 0: Data Event 1: Call Status Event 2: Web Collab Status Event 3: Failover Event
eventUri	All except GET /event	This encoded URI is returned from multiple REST APIs. It can be used to post new data in a collaboration session or to poll an event, such as a change in call status, change in collaboration status, or new data becoming available in a collaboration session. It doesn't return immediately. It will wait for an event to occur before a response is sent, or when a configured amount of time transpires. You can configure the time from the administrative console and is called the Maximum hold time.
callerAddressOfRecord	All including /call or /callNotification in the URI	Address of record for the calling phone.
calleeAddressOfRecord	All including /call or /callNotification in the URI	Address of record for the phone that was called.
callServiceUri	All including /call or /callNotification in the URI	Encoded URI is returned from the make a call request, used for getting status on, or ending a call.
callNotifyUri	All except GET /event	This encoded URI is returned from registering for call notification, used for monitoring a phone to see if a call has arrived or to unregister call notification.

Table 13. Response fields (continued).

This table shows response fields and their descriptions.

Response field	Applicable HTTP Method and URI	Description
callId	All including /call or /callNotification in the URI when a call is active	Call ID associated with the current active call.
callFailureReason	GET /call	Upon a failed call, this is a message indicating the reason for the failure.
callStatus	GET /call	Represents the status of the call (initiated, established, failed, cleared).
collaborationStatus	All except GET /event	Represents the status of a web collaboration session. Valid states are as follows: ESTABLISHED, NOT_ESTABLISHED, STARTING, and READY.
collaborationServiceUri	All except GET /event when collaboration is enabled	This encoded URI is returned from the enable collaboration request, used for getting status or ending a web collaboration session.
collaborationId	All except GET /event when collaboration is enabled	Unique identifier of a user collaboration session. If an optional address of record is provided when enabling collaboration, then the collaborationId matches that value.
forPeerCollaborationUri	All except GET /event when collaboration is enabled	URI that a user can send to a peer. The peer uses it to establish a web collaboration session. It includes the peerAddressOfRecord parameter.
peerCollaborationUri	All except GET /event when collaboration is enabled and the peer in the collaboration session has been found	This encoded URI can be used to establish a web collaboration with the other user on the phone call.
ceaVersion	All	This string represents the version of the server.

Return codes

The following table lists the possible return codes seen in the REST response returnCode field.

Table 14. Return codes.

This table shows possible return codes and their descriptions.

Code number	Description
200	Similar to an HTTP 200 OK. No problems were detected.
201	Response to the GET /event request if no new events are found. This return does not imply an error because an event might not have been generated.
300	REST request was invalid. For example, the HTTP method and URI did not match what is supported by the API.

Table 14. Return codes (continued).

This table shows possible return codes and their descriptions.

Code number	Description
301	REST request was a follow-on request, such as getting call or collaboration status, but no existing session information was found.
302	An error occurred when trying to parse the REST request. An error exists in the format of the JSON or XML provided.
303	For telephony-related REST requests, a user name was not found in the credentials. This would only happen if the configuration indicates that the user name associated with a caller should be retrieved from the request credentials.
304	Response to an outstanding GET /event request if, before a response is sent, a redundant GET /event request is received.
305	REST request is missing required parameters or fields. For example, this return results if the API makes a call that was sent without providing an addressOfRecord identifying the caller.
306	An unexpected error occurred.
307	A telephony related REST request was made that did not locate a call.
308	A collaboration-related REST request was made, but collaboration was not enabled for in the session.

Sample REST requests and responses

Enabling collaboration with JSON

```
REST Request:
   PUT http://host:port/commsvc.rest/CommServlet/collaborationSession?JSON=true&ceaVersion=1.0.0.1

REST Response:
   {
   "returnCode":200,
   "infoMsg":"Successfully enabled collaboration",
   "collaborationId":"local.1242138965934_1",
   "callNotifyUri":"CommServlet/callerNotification;ibmappid=local.1242138965934_1",
   "collaborationStatus":"NOT_ESTABLISHED",
   "collaborationServiceUri":"CommServlet/collaborationSession;ibmappid=local.1242138965934_1",
   "forPeerCollaborationUri":"CommServlet/collaborationSession?addressOfRecord=local.1242138965934_1",
   "eventUri":"CommServlet/event;ibmappid=local.1242138965934_1"
   "ceaVersion":"1.0.0.1"
   }
```

Enabling collaboration with XML

```
REST Request:
   PUT http://host:port/commsvc.rest/CommServlet/collaborationSession?ceaVersion=1.0.0.1

REST Response:
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <CommRestResponse xmlns="http://jaxb.servlet.commsvc.ws.ibm.com/">
      <returnCode>200</returnCode>
      <infoMsg>Successfully enabled collaboration</infoMsg>
      <callNotifyUri>CommServlet/callerNotification;ibmappid=local.1242140626552_1</callNotifyUri>
      <collaborationId>local.1242140626552_1</collaborationId>
      <collaborationStatus>NOT_ESTABLISHED</collaborationStatus>
      <collaborationServiceUri>CommServlet/collaborationSession;ibmappid=local.1242140626552_1</collaborationServiceUri>
      <forPeerCollaborationUri>CommServlet/collaborationSession?addressOfRecord=local.1242140626552_1</forPeerCollaborationUri>
      <eventUri>CommServlet/event;ibmappid=local.1242140626552_1</eventUri>
      <ceaVersion>1.0.0.1</ceaVersion>
   </CommRestResponse>
```

Making a call with JSON

```
REST Request
    PUT http://host:port/commsvc.rest/CommServlet/call?JSON=true&ceaVersion=1.0.0.1
    {
    "enableCollaboration": false,
    "addressOfRecord":"sip:phone1@192.168.1.100",
    "peerAddressOfRecord":"sip:phone2@192.168.1.100"
    }

 REST Response:
    {
    "returnCode":200,
    "infoMsg":"Call attempted between sip:phone1@192.168.1.100 and sip:phone2@192.168.1.100.",
    "callerAddressOfRecord":"sip:phone1@192.168.1.100",
    "calleeAddressOfRecord":"sip:phone2@192.168.1.100",
    "callServiceUri":"CommServlet/call;ibmappid=local.1242140626552_42",
    "callNotifyUri":"CommServlet/callerNotification;ibmappid=local.1242140626552_42",
    "collaborationStatus":"NOT_ESTABLISHED",
    "eventUri":"CommServlet/event;ibmappid=local.1242140626552_42"
    "ceaVersion":"1.0.0.1"
    }
```

Making a call with XML

```
 REST Request
     PUT http://host:port/commsvc.rest/CommServlet/call?ceaVersion=1.0.0.1
     <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <CommRestRequest xmlns="http://jaxb.servlet.commsvc.ws.ibm.com/">
        <addressOfRecord>sip:phone1@192.168.1.100</addressOfRecord>
        <peerAddressOfRecord>sip:phone2@192.168.1.100</peerAddressOfRecord>
        <enableCollaboration>true</enableCollaboration>
     </CommRestRequest>

 REST Response
     <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <CommRestResponse xmlns="http://jaxb.servlet.commsvc.ws.ibm.com/">
        <returnCode>200</returnCode>
        <infoMsg>Call attempted between sip:phone1@192.168.1.100 and sip:phone2@192.168.1.100.</infoMsg>
        <callerAddressOfRecord>sip:phone1@192.168.1.100</callerAddressOfRecord>
        <calleeAddressOfRecord>sip:phone2@192.168.1.100</calleeAddressOfRecord>
        <callServiceUri>CommServlet/call;ibmappid=local.1242140626552_33</callServiceUri>
        <callNotifyUri>CommServlet/callerNotification;ibmappid=local.1242140626552_33</callNotifyUri>
        <collaborationStatus>NOT_ESTABLISHED</collaborationStatus>
        <eventUri>CommServlet/event;ibmappid=local.1242140626552_33</eventUri>
        <ceaVersion>1.0.0.1</ceaVersion>
     </CommRestResponse>
```

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

IBM i

Default product locations - IBM i

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations.

app_client_root

The default installation root directory for the Application Client for IBM WebSphere Application Server is the `/QIBM/ProdData/WebSphere/AppClient/V85/client` directory.

app_client_user_data_root

The default Application Client for IBM WebSphere Application Server user data root is the `/QIBM/UserData/WebSphere/AppClient/V85/client` directory.

app_client_profile_root

The default Application Client for IBM WebSphere Application Server profile root is the `/QIBM/UserData/WebSphere/AppClient/V85/client/profiles/`*`profile_name`* directory.

app_server_root

The default installation root directory for WebSphere Application Server is the `/QIBM/ProdData/WebSphere/AppServer/V85/Base` directory.

java_home

Table 15. Root directories for supported Java Virtual Machines.

This table shows the root directories for all supported Java Virtual Machines (JVMs).

JVM	Directory
32–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit
64–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit

plugins_profile_root

The default Web Server Plug-ins profile root is the `/QIBM/UserData/WebSphere/Plugins/V85/webserver/profiles/`*profile_name* directory.

plugins_root

The default installation root directory for Web Server Plug-ins is the `/QIBM/ProdData/WebSphere/Plugins/V85/webserver` directory.

plugins_user_data_root

The default Web Server Plug-ins user data root is the `/QIBM/UserData/WebSphere/Plugins/V85/webserver` directory.

product_library
product_lib

This is the product library for the installed product. The product library for each Version 8.5 installation on the system contains the program and service program objects (similar to `.exe`, `.dll`, `.so` objects) for the installed product. The product library name is `QWAS85`*x* (where *x* is A, B, C, and so on). The product library for the first WebSphere Application Server Version 8.5 product installed on the system is `QWAS85A`. The *app_server_root*`/properties/product.properties` file contains the value for the product library of the installation, `was.install.library`, and is located under the *app_server_root* directory.

profile_root

The default directory for a profile named *profile_name* for WebSphere Application Server is the `/QIBM/UserData/WebSphere/AppServer/V85/Base/profiles/`*profile_name* directory.

shared_product_library

The shared product library, which contains all of the objects shared by all installations on the system, is `QWAS85`. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

user_data_root

The default user data directory for WebSphere Application Server is the `/QIBM/UserData/WebSphere/AppServer/V85/Base` directory.

The `profiles` and `profileRegistry` subdirectories are created under this directory when you install the product.
The *user_data_root* directory contains the default locations for `WLP_USR_DIR` and `WLP_OUTPUT_DIR` when the Liberty profile is installed. These directories are *user_data_root*`/wlp/usr` and *user_data_root*`/wlp/output/servers`, respectively.

web_server_root

The default web server path is `/www/`*web_server_name*.

Chapter 7. Client applications

This page provides a starting point for finding information about application clients and client applications. Application clients provide a framework on which application code runs, so that your client applications can access information on the application server.

For example, an insurance company can use application clients to help offload work on the server and to perform specific tasks. Suppose an insurance agent wants to access and compile daily reports. The reports are based on insurance rates that are located on the server. The agent can use application clients to access the application server where the insurance rates are located. More introduction...

Types of client applications

You can write client applications that run separately from your application server. A client application uses the framework provided by an underlying client to access the resources provided by WebSphere Application Server.

Several types of clients are installed either with WebSphere Application Server or, optionally, with the Application Client for WebSphere Application Server.

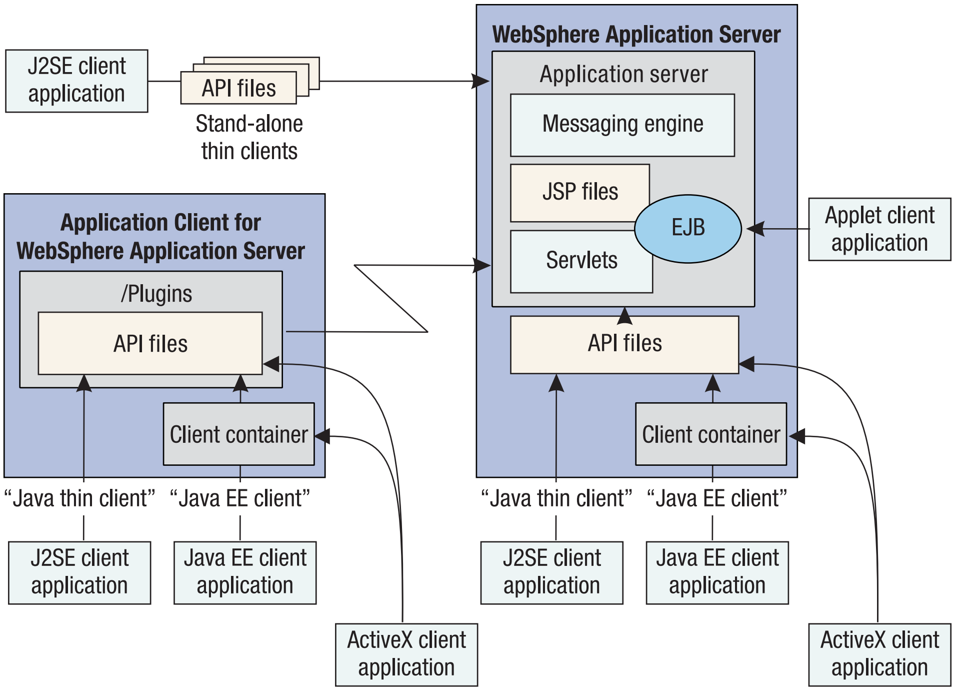


Figure 9. Clients provided for WebSphere Application Server

Stand-alone thin clients and resource adapter for JMS
The stand-alone thin clients are small, embeddable Java SE clients that you can run either on their own or, to provide different features, with one or more other stand-alone thin clients. The resource adapter for JMS is a stand-alone resource adapter that provides third party application servers with full connectivity to service integration resources running inside WebSphere Application Server.

The following table provides a comparison of the different types of clients that are available.

Table 16. Client comparison. The first column of this table lists the attributes that are being compared, and the remaining seven columns show the comparison information for each of the different types of client that are available.

	Stand-alone thin clients	**Resource Adapter for JMS**	**Java EE client (Java EE mode of Application Client)**	**Java thin client (Java SE mode of Application Client)**	**Java EE client (Java EE mode of Application Server[1]**	**Java thin client (Java SE mode of Application Server) [1]**	**Pluggable Application Client (deprecated)**
Unique characteristics	Embeddable single jar with small footprint	JCA v1.5 resource adapter with small footprint	Large client footprint with many files	Large client footprint with many files	Very large server footprint with many files	Very large server footprint with many files	Large client footprint with many files (subset of Application Client for WebSphere Application Server
Supported execution environment	Java SE	Java EE v1.4 application server: Apache Geronimo, WebSphere Application Server Community Edition, JBoss	Java EE client container	Java SE	Java EE client container	Java SE	Java SE
Supported Java vendors	IBM, Sun, and HP-UX	As per J2EE vendor	Supplied IBM application server	Supplied IBM application server	Supplied IBM application server	Supplied IBM application server	Sun
Supported Java version	See Table 17 on page 75	• 1.5 • 1.6	Supplied IBM JRE	Supplied IBM JRE	Supplied IBM JRE	Supplied IBM JRE	1.5+
Supported transactions	No transactions and local transactions	No transactions, local transactions, and XA transactions for JMS	No transactions, and local transactions for JMS	No transactions, and local transactions for JMS	No transactions, local transactions for JMS	No transactions, and local transactions for JMS	No transactions, and local transactions for JMS
Easily embedded	Yes	No	No	No	No	No	No
Include JNDI lookup capability to WebSphere Application Server	Available through the Thin Client for Enterprise JavaBeans (EJB)	Not applicable (relies on host application server JNDI)	Yes	Yes	Yes	Yes	Yes
Connectivity support	TCP and SSL	TCP and SSL	TCP, HTTP, and SSL	TCP, HTTP, and SSL	TCP, HTTP, and SSL	TCP, HTTP, and SSL	TCP and HTTP
Notable restrictions	Thin Client for JMS does not support HTTP connectivity. For web services, the use of SOAP/JMS is not supported by the thin client environment	No HTTP connectivity	None	None	None	None	No SSL support

Table 16. Client comparison (continued). The first column of this table lists the attributes that are being compared, and the remaining seven columns show the comparison information for each of the different types of client that are available.

	Stand-alone thin clients	Resource Adapter for JMS	Java EE client (Java EE mode of Application Client)	Java thin client (Java SE mode of Application Client)	Java EE client (Java EE mode of Application Server[1]	Java thin client (Java SE mode of Application Server) [1]	Pluggable Application Client (deprecated)
License type	IPLA (unlimited copy but no redistribution), and ILAN (redistribution)	IPLA (unlimited copy but no redistribution), and ILAN (redistribution)	IPLA (unlimited copy but no redistribution), and ILAN (redistribution)	IPLA (unlimited copy but no redistribution), and ILAN (redistribution)	IPLA	IPLA	IPLA (unlimited copy but no redistribution), and ILAN (redistribution)

[1] The information in this column relates to WebSphere Application Server when used as the client runtime environment.

The following table provides additional information on the supported JRE versions for stand-alone thin clients.

Table 17. Supported JRE versions. The first column of this table lists the stand-alone thin clients, and the second column lists the supported JRE versions for each of the stand-alone thin clients.

Type	JRE Versions
Enterprise JavaBeans thin client	• 1.5 • 1.6 • 1.7
Java Message Service thin client	• 1.5 • 1.6 • 1.7
Java API for XML-based RPC (JAX-RPC) thin client	• 1.6 • 1.7
Java API for XML-Based Web Services (JAX-WS) thin client	• 1.6 • 1.7
Administrative thin client	• 1.6 • 1.7
Java Persistence API (JPA)	• 1.5 • 1.6 • 1.7

Terms used for clients

Clients provided by WebSphere Application Server, and client applications that you develop, are referred to by similar terms. The terms described in this topic should help you better understand other client-related information.

Application Client

Application Client for WebSphere Application Server is the package that you can use to install a variety of clients.

Application Client also forms the runtime for Java EE clients and Java thin clients on a system that does not have the Application Server installed.

client Provides a framework on which an application runs, so that the application can access information on an application server. Clients are provided as part of the Application Client for WebSphere Application Server or as part of a WebSphere Application Server installation.

Clients are sometimes referred to as “application clients”.

client application

The application program that you develop to access information on an application server. The application is built on the framework provided by one or more clients.

Java EE client

The Java Platform, Enterprise Edition (Java EE) client is a Java EE mode of using the runtime environment of either an Application Client installation or a WebSphere Application Server installation. The Java EE client uses the Client Container in the runtime environment to simplify access to system services such as security, transactions, naming, and database access for use by Java EE client applications.

The Java EE client is sometimes referred to as the “Java EE application client” or “J2EE application client”.

Java thin client

The Java thin client is a JavaPlatform, Standard Edition (Java SE) mode of using the runtime environment of either an Application Client installation or a WebSphere Application Server installation. The Java thin client runtime environment provides the support needed by full-function Java SE client applications for object resolution, security, Reliability Availability and Servicability (RAS), and other services. However, the Java thin client does not support a Client Container that provides easy access to these services.

The Java thin client is sometimes referred to as the “Java thin application client”.

stand-alone thin client

Small embeddable Java SE clients that you can use either on their own or, to provide different features, with one or more other stand-alone thin clients. Stand-alone thin clients are provided as embeddable JAR files, and have names such as “IBM Thin Client for *feature*”; for example, “IBM Thin Client for Java Messaging Service (JMS)”.

Application Client for WebSphere Application Server

Application Client for WebSphere Application Server is the package that you can use to install a variety of clients. Application Client also forms the runtime for Java EE clients and Java thin clients on a system that does not have the Application Server installed.

The Application Client for WebSphere Application Server is packaged with the following components:

- IBM i The Java extensions needed to work with the IBM i server Java Development Kit. The IBM i product does not include a stand-alone JRE.
- The runtime environment for Java EE client applications (that use services provided by the Java EE Client Container)
- The runtime environment for Java thin client applications (Java SE applications that do not use services provided by the Java EE Client Container)
- A variety of stand-alone thin clients, as embeddable JAR files

Stand-alone thin clients

Small embeddable Java SE clients that you can use either on their own (stand-alone) or, to provide different features, with one or more other stand-alone thin clients.

The stand-alone thin clients are provided as embeddable JAR files in the %WAS_HOME%/runtimes directory of either an Application Client installation or a WebSphere Application Server installation.

IBM Thin Client for Java Messaging Service (JMS)

The Thin Client for JMS is a Java service integration bus JMS client designed to run as an embeddable client in Java SE applications under the IBM, Sun and HP Java run-time environments (JREs). The client supports no transaction and local transaction models.

IBM Thin Client for Enterprise JavaBeans (EJB)
The Thin Client for EJB allows Java SE applications to access remote Enterprise Java Beans on a server through Java Naming and Directory Interface (JNDI) look up. It can be embedded in a Java SE application running under the IBM, Sun, or HP JREs.

IBM Thin Client for Java API for XML-based Web Services (JAX-WS)
The Thin Client for JAX-WS allows Java SE client applications to use JAX-WS to invoke web services that are hosted by an application server. Such unmanaged client applications can use JAX-WS APIs to directly inspect a WSDL file and formulate the calls to web services

IBM Thin Client for Java API for XML-based RPC (JAX-RPC)
The Thin Client for JAX-RPC allows Java SE client applications to use JAX-RPC to invoke web services that are hosted by an application server. Such unmanaged client applications can access a web service as if the web service is a local object mapped into the client address space even though the web service provider is located in another part of the world.

IBM Thin Client for Java API for RESTful Web Services (JAX-RS)
The Thin Client for JAX-RS is a stand-alone Java SE 6 client environment that enables running unmanaged JAX-RS RESTful web services client applications in a non-WebSphere environment to invoke JAX-RS RESTful web services that are hosted by the application server.

IBM Thin Client for Java Persistence API (JPA)
The Thin Client for JPA allows Java SE client applications to use the Java Persistence API (JPA) to store and retrieve persistent data without the use of an application server.

If you are running two or more of these stand-alone thin clients together, you must obtain all the clients that you are using from the same installation of Application Client for WebSphere Application Server, the same installation of the WebSphere Application Server product, or the same service refresh.

Although the stand-alone thin clients can coexist with each other, none of them can coexist with the Administration Thin Client for WebSphere Application Server.

WebSphere Application Server and IBM Application Client for WebSphere Application Serveralso provide a resource adapter for JMS that enables a third-party application server to be a stand-alone JMS client of WebSphere Application Server:

IBM Resource Adapter for JMS with WebSphere Application Server
A Java EE Connector Architecture (JCA) V1.5-compliant resource adapter that runs in a supported Java EE V1.4 compliant application server. The resource adapter provides full two-phase transaction support through an XA interface, supports inbound messages through message-driven beans (MDBs) and supports connection pooling with lazy association.

Java EE client

The Java Platform, Enterprise Edition (Java EE) client is a Java EE mode of using the runtime environment of either an Application Client installation or a WebSphere Application Server installation. The Java EE client uses the Client Container in the runtime environment to simplify access to system services such as security, transactions, naming, and database access for use by Java EE client applications.

The Java EE client is sometimes referred to as the “Java EE application client” or “J2EE application client”.

The Client Container enables Java EE client applications to use logical names (“nicknames”) for enterprise beans and local resources, and to leave the resolution of those names to a look up in the Java Naming and Directory Interface (JNDI) namespace of an application server. Besides simplifying resolution to enterprise beans and local resources references, this use of logical names and JNDI lookups eliminates changes to the client application code if the underlying object or resource either changes or moves to a different application server.

The Java EE client initializes the runtime environment for a Java EE client application. A deployment descriptor defines the unique initialization for a client application, and defines the logical names used by the application.

The logical names are defined within the deployment descriptor of a Java EE client application. These logical deployment descriptors identify enterprise beans or local resources (Java Database Connectivity (JDBC) data sources, J2C connection factories, Java Message Service (JMS) resources, and JavaMail and URL APIs) for simplified resolution through JNDI lookup.

Storing the resource information separately from the client application program makes the client application program portable and more flexible. If you develop a client application using and adhering to the Java EE platform, you can port the client application from one Java EE platform implementation to another. The code of the client application does not change, but the application package might need redeployment using the deployment tool of the new Java EE platform.

Attention: The Java EE client does not support connection pools. The application client calls the database directly, without a datasource. If you want to use the getConnection() request from a Java EE client application, configure the JDBC provider in the application deployment descriptors, using Rational® Application Developer or an assembly tool. The connection is established between the client application and the database.

The Java EE client uses the Java Remote Method Invocation technology over Internet Inter-Orb Protocol (RMI-IIOP). Using this protocol enables a Java EE client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the Java EE client runtime. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a Java EE client application that requires access to both enterprise bean references and CORBA object references.

Java thin client

The Java thin client is a JavaPlatform, Standard Edition (Java SE) mode of using the runtime environment of either an Application Client installation or a WebSphere Application Server installation. The Java thin client runtime environment provides the support needed by full-function Java SE client applications for object resolution, security, Reliability Availability and Serviceability (RAS), and other services. However, the Java thin client does not support a Client Container that provides easy access to these services.

The Java thin client is sometimes referred to as the “Java thin application client”.

The Java thin client is designed to support those users who want a full-function Java SE client application programming environment, to use the supplied IBM JRE, without the overhead of the Java Platform, Enterprise Edition (Java EE) platform on the client machine.

The Java thin client does not perform initialization of any of the services that the client application might require. For example, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI APIs.

The Java thin client does not support the use of use of logical names (“nicknames”) for enterprise beans and local resources. When a client application resolves a reference for an enterprise bean (using either Java Naming and Directory Interface (JNDI) or CosNaming), the application must know the location of the name server and the fully-qualified name used when the reference was bound into the name space. When a client application resolves a reference for a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API; for example, JDBC or Java Message Service (JMS). If the location of an enterprise bean or resource changes, the thin client application must also change the value placed on the lookup() statement.

The Java thin client runtime environment provides support for Java SE client applications to access remote enterprise beans, and provides the implementation for various enterprise bean services. Client applications can also use the Java thin client runtime environment to access CORBA objects and CORBA based services.

The Java thin client uses the RMI-IIOP protocol, which enables the client application to access both enterprise bean references and CORBA object references. Using this protocol also allows the client application to use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can help you develop a client application that needs to access both enterprise bean references and CORBA object references.

If you choose to use both enterprise beans and CORBA programming models in the same client application, you need to understand the differences between those programming models to manage both environments. For example, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model, for which the JNDI implementation initializes the Object Request Broker (ORB); the client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the ORB.init() static method.

Note: The CORBA model does not allow for workload management (WLM) functionality and cluster failover. Use the enterprise bean model (with JNDI) to access objects in a clustered environment.

The Java thin application client provides a batch command that you can use to set the CLASSPATH and JAVA_HOME environment variables to enable the Java thin application client run time.

gotcha: When running in an environment that includes Java thin application clients, clients might suddenly encounter a situation where the port information about the cluster members of the target cluster has become stale. This situation most commonly occurs when all of the cluster members have dynamic ports and are restarted during a time period when no requests are being sent. The client process in this state will eventually attempt to route to the node agent to receive the new port data for the cluster members, and then use that new port data to route back to the members of the cluster.

If any issues occur that prevent the client from communicating with the node agent, or that prevent the new port data being propagated between the cluster members and the node agent, request failures might occur on the client. In some cases, these failures are temporary. In other cases you need to restart one or more processes to resolve a failure.

To circumvent the client routing problems that might arise in these cases, you can configure static ports on the cluster members. With static ports, the port data does not change as a client process gets information about the cluster members. Even if the cluster members are restarted, or there are communication or data propagation issues between processes, the port data the client holds is still valid. This circumvention does not necessarily solve the underlying communication or data propagation issues, but removes the symptoms of unexpected or uneven client routing decisions.

Applet client

The Applet client provides a browser-based Java run time capable of interacting with enterprise beans directly, instead of indirectly through a servlet.

This client is designed to support users who want a browser-based Java client application programming environment that provides a richer and more robust environment than the one offered by the **Applet > Servlet > enterprise bean** model.

The programming model for this client is a hybrid of the Java application thin client and a servlet client. When accessing enterprise beans from this client, the applet can consider the enterprise bean object references as CORBA object references.

No tooling support exists for this client to develop, assemble or deploy the applet. You are responsible for developing the applet, generating the necessary client bindings for the enterprise beans and CORBA objects, and bundling these pieces together to install or download to the client machine. The Java applet client provides the necessary run time to support communication between the client and the server. The applet client run time is provided through the Java applet browser plug-in that you install on the client machine.

Generate client-side bindings using an assembly tool. An applet can utilize these bindings, or you can generate client-side bindings using the **rmic** command. This command is part of the IBM Developer Kit, Java edition that is installed with the WebSphere Application Server.

The applet client uses the RMI-IIOP protocol. Using this protocol enables the applet to access enterprise bean references and CORBA object references, but the applet is restricted in using some supported CORBA services.

If you combine the enterprise bean and CORBA environments in one applet, you must understand the differences between the two programming models, and you must use and manage each model appropriately.

The applet environment restricts access to external resources from the browser runtime environment. You can make some of these resources available to the applet by setting the correct security policy settings in the WebSphere Application Server `client.policy` file. If given the correct set of permissions, the applet client must explicitly create the connection to the resource using the appropriate API. This client does not perform initialization of any service that the client applet can need. For example, the client application is responsible for the initialization of the naming service, either through the CosNaming, or the Java Naming and Directory Interface (JNDI) APIs.

ActiveX to Enterprise JavaBeans (EJB) Bridge

WebSphere Application Server provides an ActiveX to EJB bridge that enables ActiveX programs to access enterprise beans through a set of ActiveX automation objects.

The bridge accomplishes this access by loading the Java virtual machine (JVM) into any ActiveX automation container such as Visual Basic, VBScript, and Active Server Pages (ASP).

There are two main environments in which the ActiveX to EJB bridge runs:

- **Client applications**, such as Visual Basic and VBScript, are programs that a user starts from the command line, desktop icon, or Start menu shortcut.
- **Client services**, such as Active Server Pages, are programs started by some automated means like the Services control panel applet.

The ActiveX to EJB bridge uses the Java Native Interface (JNI) architecture to programmatically access the JVM code. Therefore the JVM code exists in the same process space as the ActiveX application

(Visual Basic, VBScript, or ASP) and remains attached to the process until that process terminates. To create JVM code, an ActiveX client program calls the XJBInit() method of the XJB.JClassFactory object.

After an ActiveX client program has initialized the JVM code, the program calls several methods to create a proxy object for the Java class. When accessing a Java class or object, the real Java object exists in the JVM code; the automation container contains the proxy for that Java object. The ActiveX program can use the proxy object to access the Java class, object fields, and methods.

To convert primitive data types, the client program uses the COM IDispatch interface (use of the IUnknown interface is not directly supported). Primitive data types are automatically converted between native automation types and Java types. All other types are handled automatically by the proxy objects.

Any exceptions thrown in Java code are encapsulated and thrown again as a COM error, from which the ActiveX program can determine the actual Java exceptions.

The ActiveX to EJB bridge supports both free-threaded and apartment-threaded access and implements the free threaded marshaler (FTM) to work in a hybrid environment such as Active Server Pages.

For more information about ActiveX client programming with the ActiveX to EJB bridge, refer to the Developing ActiveX client application code topic.

Pluggable Application Client

The Pluggable Application Client for WebSphere Application Server provides a downloadable run time for Java client applications to run with the Sun Java Runtime Environment (JRE) on the Windows platform.

Important: The Pluggable Application Client is deprecated. It is replaced by the stand-alone thin client, IBM Thin Client for EJB.

The Pluggable Application Client runs only on the Windows platform and requires that you have previously installed the Sun Java Runtime Environment (JRE) files. In all other aspects, the Pluggable Application Client and the Java thin application client are similar.

Chapter 8. Data access resources

This page provides a starting point for finding information about data access. Various enterprise information systems (EIS) use different methods for storing data. These backend data stores might be relational databases, procedural transaction programs, or object-oriented databases.

The flexible IBM WebSphere Application Server provides several options for accessing an information system backend data store:
- Programming directly to the database through the JDBC 4.0 API, JDBC 3.0 API, or JDBC 2.0 optional package API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA-compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

Service Data Objects (SDO) simplify the programmer experience with a universal abstraction for messages and data, whether the programmer thinks of data in terms of XML documents or Java objects. For programmers, SDOs eliminate the complexity of the underlying data access technology such as JDBC, RMI/IIOP, JAX-RPC, and JMS, and message transport technology such as, java.io.Serializable, DOM Objects, SOAP, and JMS.

Data concepts

Relational resource adapters and JCA

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

WebSphere Application Server supports JCA versions 1.0, 1.5 and 1.6, including additional configurable features for JCA 1.5 resource adapters with activation specifications that handle inbound requests. The JCA Version 1.6 specification also adds support for Java annotations in RAR modules. For more information on annotation support see the topic, JCA 1.6 support for annotations in RAR modules.

Data access for container-managed persistence (CMP) beans is indirectly managed by the WebSphere Persistence Manager. The JCA specification supports persistence manager delegation of the data access to the JCA resource adapter without knowing the specific backend store. For the relational database access, the persistence manager uses the relational resource adapter to access the data from the database.

You can find the supported database platforms for the JDBC API at the WebSphere Application Server prerequisite website.

Java EE Connector Architecture and WebSphere relational resource adapters

An application server vendor extends its system once to support the Java Platform, Enteprise Edition Connector Architecture (JCA) and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter with the capability to plug into any application server that supports the connector architecture.

The product supports any resource adapter that implements version 1.0, 1.5 and 1.6 of this specification. IBM includes WebSphere MQ and the Service Integration Bus with the Application Server, and IBM supplies resource adapters for many enterprise systems separately from the WebSphere Application Server package, which include but are not limited to, the Customer Information Control System (CICS®), Host On-Demand (HOD), Information Management System (IMS™), and Systems, Applications, and Products (SAP) R/3 .

The general approach to writing an application that uses a JCA resource adapter is to develop EJB session beans or services with tools such as Rational Application Developer. The session bean uses the *javax.resource.cci* interfaces to communicate with an enterprise information system through the resource adapter.

WebSphere Relational Resource Adapter

WebSphere Application Server provides the WebSphere Relational Resource Adapter implementation. This resource adapter provides data access through JDBC calls to access the database dynamically. The connection management is based on the JCA connection management architecture and provides connection pooling, transaction, and security support. The WebSphere RRA is installed and runs as part of WebSphere Application Server, and needs no further administration.

The RRA supports both the configuration and use of JDBC data sources and JCA connection factories. The RRA supports the configuration and use of data sources implemented as either JDBC data sources or Java EE Connector Architecture connection factories. Data sources can be used directly by applications, or they can be configured for use by container-managed persistence (CMP) entity beans.

For more information about the WebSphere Relational Resource Adapter, see the following topics:

- For information about resource adapters and data access, see the topic Data access portability features.
- For RRA settings, see the topic WebSphere relational resource adapter settings.
- For information about enterprise beans, see the topic EJB applications.

Using a single instance of a resource adapter

You can restrict certain resource adapters to a single runtime instance inside the Java Virtual Machine (JVM).

Before you begin

Enabling this setting imposes a highly restrictive environment on the system and should be used with caution.

About this task

Using the single-instance resource adapter configuration option on some resource adapters can enable you to set up an environment that optimally behaves. Some resource adapters that support inbound communications from the enterprise information system (EIS) might require single-instance behavior. By enabling this setting, server startup time can be optimized. Other resource adapters might not require this setting. You need to determine if you should configure the resource adapter for single-instance behavior.

Consider using the single-instance resource adapter configuration for testing and troubleshooting problems. Placing the single-instance restriction on some resource adapters might work as a corrective action for problems; enabling single-instance behavior on one or more resource adapters thought to be involved in a problem can help isolate the specific issue.

This design does not allow two resource adapter JavaBeans instances that would return true from the equals method to coexist in the same JVM, if any one of them is configured as single-instance. For example, if two applications that have embedded the same resource adapter, or one application that embeds a resource adapter and the same resource adapter is installed in the server as a stand-alone resource adapter, are configured on the same server such that even though some of their config attributes are different, the ones that the equals() method evaluates are equal, this will no longer be allowed, and will return a ResourceException.

Note: The vendor of a resource adapter which cannot tolerate multiple instances does not have a JCA-defined method of communicating this. Therefore, it is up to the deployer to recognize the need, and configure resource adapter(s) for single-instance behavior.

WebSphere relational resource adapter settings

Use this page to view the settings of the WebSphere relational resource adapter. This adapter is preinstalled in the product to provide access to relational databases.

Restriction: Although the default relational resource adapter settings are viewable, you cannot make changes to them.

To view this administrative console page, click **Resources** > **Resource adapters** > **Resource adapters**. Expand the **Preferences** section at the top of the page. Select **Show built-in resources**. The table of configured resource adapters now displays the **WebSphere Relational Resource Adapter**.

Name:

Specifies the name of the resource provider.

Information	Value
Data type	String

Description:

Specifies a description of the relational resource adapter.

Information	Value
Data type	String

Scope:

Specifies the scope of the relational resource adapter.

Information	Value
Data type	String

Data access portability features

These interfaces work with the relational resource adapter (RRA) to make database-specific functions operable on connections between the application server and that database.

In other words, your applications can access data from different databases, and use functions that are specific to the database, without any code changes. Additionally, WebSphere Application Server enables you to plug in a data source that is not supported by WebSphere persistence. However, the data source *must* be implemented as either the *XADataSource* type or the *ConnectionPoolDataSource* type, and it must be in compliance with the JDBC 2.x specification.

You can achieve application portability through the following:

DataStoreHelper interface

With this interface, each data store platform can plug in its own private datastore specific functions that the relational resource adapter runtime uses. WebSphere Application Server provides an implementation for each supported JDBC provider.

The interface also provides a GenericDataStoreHelper class for unsupported data sources to use. You can subclass the GenericDataStoreHelper class or other WebSphere provided helpers to support any new data source.

Note: If you are configuring data access through a user-defined JDBC provider, do not implement the DataStoreHelper interface directly. Either subclass the GenericDataStoreHelper class or subclass one of the DataStoreHelper implementation classes provided by IBM (if your database behavior or SQL syntax is similar to one of these provided classes).

For more information, see the API documentation **DataStoreHelper** topic (as listed in the API documentation index).

The following code segment shows how a new data store helper is created to add new error mappings for an unsupported data source.

```
public class NewDSHelper extends GenericDataStoreHelper
{
  public NewDSHelper(java.util.Properties dataStoreHelperProperties)
  {
    super(dataStoreHelperProperties);
    java.util.Hashtable myErrorMap = null;
    myErrorMap = new java.util.Hashtable();
    myErrorMap.put(new Integer(-803), myDuplicateKeyException.class);
    myErrorMap.put(new Integer(-1015), myStaleConnectionException.class);
    myErrorMap.put("S1000", MyTableNotFoundException.class);
    setUserDefinedMap(myErrorMap);
    ...
  }
}
```

WSCallHelper class

This class provides two methods that enable you to use vendor-specific methods and classes that do not conform to the standard JDBC APIs (and are not part of WebSphere Application Server extension packages).

- **jdbcCall() method**

  By using the static jdbcCall() method, you can invoke vendor-specific, nonstandard JDBC methods on your JDBC objects. (For more information, see the API documentation **WSCallHelper** topic.) The following code segment illustrates using this method with a DB2® data source:

  ```
  Connection conn = ds.getConnection();
  // get connection attribute
  String connectionAttribute =(String) WSCallHelper.jdbcCall(DataSource.class, ds,
   "getConnectionAttribute", null, null);
  // setAutoClose to false
  WSCallHelper.jdbcCall(java.sql.Connection.class,
  conn, "setAutoClose",
  new Object[] { new Boolean(false)},
  new Class[] { boolean.class });
  // get data store helper
  DataStoreHelper dshelper = WSCallHelper.getDataStoreHelper(ds);
  ```

- **jdbcPass() method**

  Use this method to exploit the nonstandard JDBC classes that some database vendors provide. These classes contain methods that require vendors' proprietary JDBC objects to be passed as parameters.

  In particular, implementations of Oracle can involve use of nonstandard classes furnished by the vendor. Methods contained within these classes include:

  ```
  oracle.sql.ArrayDescriptor ArrayDescriptor.createDescriptor(java.lang.String, java.sql.Connection)
  oracle.sql.ARRAY new ARRAY(oracle.sql.ArrayDescriptor, java.sql.Connection, java.lang.Object)
  oracle.xml.sql.query.OracleXMLQuery(java.sql.Connection, java.lang.String)
  ```

```
oracle.sql.BLOB.createTemporary(java.sql.Connection, boolean, int)
oracle.sql.CLOB.createTemporary(java.sql.Connection, boolean, int)
oracle.xdb.XMLType.createXML(java.sql.Connection, java.lang.String)
```

The following code sample demonstrates how to use jdbcPass to call the Oracle method `XMLType.createXML` on a connection. This Oracle function creates an XML type object out of the XML data that the database passes to your application.

```
XMLType poXML = (XMLType)(WSCallHelper.jdbcPass(XMLType.class,
"createXML", new Object[]{conn,poString},
 new Class[]{java.sql.Connection.class, java.lang.String.class},
 new int[]{WSCallHelper.CONNECTION,WSCallHelper.IGNORE}));
```

For more examples of using jdbcPass and a complete list of method parameters, see the API documentation for the WSCallHelper class. In this information center, access the API documentation with the following steps:

1. Click **Reference** > **Developer API documentation** > **Application programming interfaces**
2. Click **com.ibm.websphere.rsadapter**
3. Under the Class Summary heading, click **WSCallHelper**

The first section on jdbcPass discusses using the method to call database static methods. The second section on jdbcPass addresses database non-static methods.

CAUTION: Use of the jdbcPass() method causes the JDBC object to be used outside of the protective mechanisms of WebSphere Application Server. Performing certain operations (such as setting autoCommit, or transaction isolation settings, etc.) outside of these protective mechanisms will cause problems with the future use of these pooled connections. IBM does not guarantee stability of the object after invocation of this method; it is the user's responsibility to ensure that invocation of this method does not perform operations that harm the object. Use at your own risk.

Because of these potential problems, WebSphere Application Server strictly controls which methods are allowed to be invoked using the jdbcPass() method support. If you require support for a method that is not listed previously in this document, contact WebSphere Application Server Support with information on the method you require.

JDBC providers

Installed applications use JDBC providers to interact with relational databases.

The JDBC provider object supplies the specific JDBC driver implementation class for access to a specific vendor database. To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

For a current list of supported providers, see the WebSphere Application Server prerequisite website. For detailed descriptions of the providers, including the supported data source classes and their required properties, refer to the topics on data source required minimum required settings, by vendor.

Configuring QueryTimeout

You can configure a timeout on the data source of an application so that the transaction or statement timeout occurs if a query fails to return, because of a deadlock or blocked transactions.

Before you begin

Traditional Java Database Connectivity (JDBC) provides a standard interface, called `java.sql.Statement.setQueryTimeout`, to limit the number of seconds that a JDBC driver waits for a statement to execute. This can used by an application to control the maximum amount of time the application waits for an SQL statement to complete before the request is interrupted. With earlier versions of WebSphere Application Server, the only way of setting a query timeout is by programmatically establishing an SQL query timeout in the application by invoking the `java.sql.Statement.setQueryTimeout` interface on every statement.

About this task

In Version 8, you can configure this query timeout using either of the following two custom properties at the data source level:

- `webSphereDefaultQueryTimeout` establishes a default query timeout, which is the number of seconds that an SQL statement may execute before timing out. This default value is overridden during a Java Transaction API (JTA) transaction if the `syncQueryTimeoutWithTransactionTimeout` custom property is enabled.
- `syncQueryTimeoutWithTransactionTimeout` uses the time remaining (if any) in a JTA transaction as the default query timeout for SQL statements.

By default, query time is disabled. Based on the presence and value of the two new data source custom properties, a timeout value is calculated of either:

- the time remaining in the current JTA transaction based on the TM timeout setting - `syncQueryTimeoutWithTransactionTimeout`
- the absolute number of seconds specified by configuration - `webSphereDefaultQueryTimeout`

The calculated timeout is then used in conjunction with the JDBC API to set a query timeout value on each statement.

Procedure

1. Open the administrative console.
2. Go to the **WebSphere Application Server Data Source properties** panel for the data source.
   a. Click **Resources** > **JDBC** > **Data Sources** > ***data_source***
   b. Click **WebSphere Application Server Data Source properties**.
3. Click **Custom properties** under Additional Properties.
4. Click **New**.
5. Enter `webSphereDefaultQueryTimeout` in the **Name** field.
6. Enter the number of seconds to use for the default query timeout in the **Value** field. The timeout value is in seconds. A value of 0 (zero) indicates no timeout.
7. Click **OK**.
8. Click **New**.
9. Enter `syncQueryTimeoutWithTransactionTimeout` in the **Name** field.
10. Enter `true` or `false` in the **Value** field. A value of `true` indicates to use the time remaining in a JTA transaction as the default query timeout.
11. Click **OK**.
12. Save your changes. The updates go into effect after the server is restarted.

Results

You have configured the query timeout on the data source of your application.

Example

The following example illustrates using `webSphereDefaultQueryTimeout = 20` and `syncQueryTimeoutWithTransactionTimeout = true`:

```
statement = connection.createStatement();
statement.executeUpdate(sqlcommand1); // query timeout of 20 seconds is used
statement.executeUpdate(sqlcommand2); // query timeout of 20 seconds is used
transaction.setTransactionTimeout(30);
transaction.begin();
try
{
    statement.executeUpdate(sqlcommand3); // query timeout of 30 seconds is used
    // assume the above operation took 5 seconds, remaining time = 30 - 5 seconds
    statement.executeUpdate(sqlcommand4); // query timeout of 25 seconds is used
    // assume the above operation took 10 seconds, , remaining time = 25 - 10 seconds
    statement.executeUpdate(sqlcommand5); // query timeout of 15 seconds is used
}
finally
{
    transaction.commit();
}
statement.executeUpdate(sqlcommand6); // query timeout of 20 seconds is used
```

The following example illustrates using `webSphereDefaultQueryTimeout = 20` and `syncQueryTimeoutWithTransactionTimeout = false`:

```
statement = connection.createStatement();
statement.executeUpdate(sqlcommand1); // query timeout of 20 seconds is used
statement.executeUpdate(sqlcommand2); // query timeout of 20 seconds is used
transaction.setTransactionTimeout(30);
transaction.begin();
try
{
    statement.executeUpdate(sqlcommand3); // query timeout of 20 seconds is used
    // assume the above operation took 5 seconds
    statement.executeUpdate(sqlcommand4); // query timeout of 20 seconds is used
    // assume the above operation took 10 seconds
    statement.executeUpdate(sqlcommand5); // query timeout of 20 seconds is used
}
finally
{
    transaction.commit();
}
statement.executeUpdate(sqlcommand6); // query timeout of 20 seconds is used
```

You can override the query timeout for a statement at any time by invoking the `java.sql.Statement.setQueryTimeout` interface from your application code.

Data sources

Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

A data source is associated with a JDBC provider, which supplies the driver implementation classes that are required for JDBC connectivity with your specific vendor database. Application components transact directly with the data source to obtain connection instances to your database. The connection pool that corresponds to each data source provides connection management.

You can create multiple data sources with different settings, and associate them with the same JDBC provider. For example, you might use multiple data sources to access different databases within the same vendor database application. WebSphere Application Server requires JDBC providers to implement one or both of the following data source interfaces, which are defined by Sun Microsystems. These interfaces enable the application to run in a single-phase or two-phase transaction protocol.

- *ConnectionPoolDataSource* - a data source that supports application participation in local and global transactions, excepting two-phase commit transactions. When a connection pool data source is involved in a global transaction, transaction recovery is not provided by the transaction manager. The application is responsible for providing the backup recovery process if multiple resource managers are involved.

  Note: A connection pool data source does support two-phase commit transactions in these cases:
  - IBM i the data source is making use of *Last participant* support. Last participant support enables a single one-phase commit resource to participate in a global transaction with one or more two-phase commit resources.

  For more information, consult the article, Using one-phase and two-phase commit resources in the same transaction.
- *XADataSource* - a data source that supports application participation in any single-phase or two-phase transaction environment. When this data source is involved in a global transaction, the product transaction manager provides transaction recovery.

Prior to version 5.0 of the application server, the function of data access was provided by a single connection manager (CM) architecture. This connection manager architecture remains available to support Java 2 Platform, Enterprise Edition (J2EE) 1.2 applications, but another connection manager architecture is provided, based on the JCA architecture supporting the J2EE 1.3 application style, J2EE 1.4 and Java EE applications.

These architectures are represented by two types of data sources. To choose the right data source, administrators must understand the nature of their applications, EJB modules, and enterprise beans.
- Data source (WebSphere Application Server V4) - This data source is utilizes the original CM architecture. Applications using this data source behave as if they were running in Version 4.0.
- Data source - This data source uses the JCA standard architecture to provide support for J2EE version 1.3 and 1.4, as well as Java EE applications. It utilizes the JCA connection manager and the relational resource adapter.

Choice of data source

- J2EE 1.2 application - all EJB 1.1 enterprise beans, JDBC applications, or Servlet 2.2 components must use the **4.0** data source.
- J2EE 1.3 (and subsequent releases) application -
  - EJB 1.1 module - all EJB 1.x beans must use the **4.0** data source.
  - EJB 2.0 (and subsequent releases) module - enterprise beans that include container-managed persistence (CMP) Version 1.x, 2.0, and beyond must use the **new** data source.
  - JDBC applications and Servlet 2.3+ components - must use the **new** data source.

Data access beans

Data access beans provide a rich set of features and function, while hiding much of the complexity associated with accessing relational databases.

They are Java classes written to the Enterprise JavaBeans specification.

You can use the data access beans in JavaBeans-compliant tools, such as the IBM *Rational Application Developer*. Because the data access beans are also Java classes, you can use them like ordinary classes.

The data access beans (in the package *com.ibm.db*) offer the following capabilities:

Feature

  Details

Caching query results

  You can retrieve SQL query results all at once and place them in a cache. Programs using the result set can move forward and backward through the cache or jump directly to any result row in the cache.

For large result sets, the data access beans provide ways to retrieve and manage *packets*, subsets of the complete result set.

Updating through result cache
: Programs can use standard Java statements (rather than SQL statements) to change, add, or delete rows in the result cache. You can propagate changes to the cache in the underlying relational table.

Querying parameter support
: The base SQL query is defined as a Java String, with parameters replacing some of the actual values. When the query runs, the data access beans provide a way to replace the parameters with values made available at run time. Default mappings for common data types are provided, but you can specify whatever your Java program and database require.

Supporting metadata
: A *StatementMetaData* object contains the base SQL query. Information about the query (*metadata*) enables the object to pass parameters into the query as Java data types.

Metadata in the object maps Java data types to SQL data types (as well as the reverse). When the query runs, the Java-datatyped parameters are automatically converted to SQL data types as specified in the metadata mapping.

When results return, the metadata object automatically converts SQL data types back into the Java data types specified in the metadata mapping.

Connection management architecture

The connection management architecture for both relational and procedural access to enterprise information systems (EIS) is based on the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification. The Connection Manager (CM), which pools and manages connections within an application server, is capable of managing connections obtained through both resource adapters (RAs) defined by the JCA specification, and data sources defined by the Java Database Connectivity (JDBC) 2.0 (and later) Extensions specification.

To make data source connections manageable by the CM, the WebSphere Application Server provides a resource adapter (the WebSphere Relational Resource Adapter) that enables JDBC data sources to be managed by the same CM that manages JCA connections. From the CM point of view, JDBC data sources and JCA connection factories look the same. Users of data sources do not experience any programmatic or behavioral differences in their applications because of the underlying JCA architecture. JDBC users still configure and use data sources according to the JDBC programming model.

Applications migrating from previous versions of WebSphere Application Server might experience some behavioral differences because of the specification changes from various Java EE requirements levels. These differences are not related to the adoption of the JCA architecture.

If you have Java 2 Platform, Enterprise Edition (J2EE) 1.2 applications using the JDBC API that you wish to run in WebSphere Application Server 6.0 and later, the JDBC CM from Application Server version 4.0 is still provided as a configuration option. Using this configuration option enables J2EE 1.2 applications to run unaltered. If you migrate a Version 4.0 application to Version 6.0 or later, using the latest migration tools, the application automatically uses the Version 4.0 connection manager after migration. However, EJB 2.x modules in J2EE 1.3, J2EE 1.4 and Java Platform, Enterprise Edition (Java EE) applications cannot use the JDBC CM from WebSphere Application Server Version 4.0.

Connection pooling

Using connection pools helps to both alleviate connection management overhead and decrease development tasks for data access.

Each time an application attempts to access a backend store (such as a database), it requires resources to create, maintain, and release a connection to that datastore. To mitigate the strain this process can place on overall application resources, the Application Server enables administrators to establish a pool of backend connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

The application server supports JDBC 4.0 APIs for connection pooling and connection reuse. The connection pool is used to direct JDBC calls within the application, as well as for enterprise beans using the database.

Benefits of connection pooling

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the web to a resource, the resource accesses a data source. Because users connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total datastore overhead quickly becomes high for Web-based applications, and performance deteriorates. When connection pooling capabilities are used, however, web applications can realize performance improvements of up to 20 times the normal results.

With connection pooling, most user requests do not incur the overhead of creating a new connection because the data source can locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. The overhead of a disconnection is avoided. Each user request incurs a fraction of the cost for connecting or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused.

When to use connection pooling

Use connection pooling in an application that meets any of the following criteria:
- It cannot tolerate the overhead of obtaining and releasing connections whenever a connection is used.
- It requires Java Transaction API (JTA) transactions within the Application Server.
- It needs to share connections among multiple users within the same transaction.
- It needs to take advantage of product features for managing local transactions within the application server.
- It does not manage the pooling of its own connections.
- It does not manage the specifics of creating a connection, such as the database name, user name, or password

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

How connections are pooled together

When you configure a unique data source or connection factory, you must give it a unique Java Naming and Directory Interface (JNDI) name. This JNDI name, along with its configuration information, is used to create the connection pool. A separate connection pool exists for each configured data source or connection factory.

Furthermore, the application server creates a separate instance of the connection pool in each application server that uses the data source or connection factory. For example:
- IBM i If you run a three server cluster in which all of the servers use *myDataSource*, and *myDataSource* has a Maximum Connections setting of 10, then you can generate up to 30 connections (three servers times 10 connections).

Consider how this behavior potentially impacts the number of connections that your backend resource can support. See the topic, Connection pool settings, for more information.

IBM i Other considerations for determining the maximum connections setting:

- Each entity bean transaction requires an additional database connection, dedicated to handling the transaction.
- If clones are used, one data pool exists for each clone.

It is also important to note that when using connection sharing, it is only possible to share connections obtained from the same connection pool.

Connection and connection pool statistics:

WebSphere Application Server supports use of PMI APIs to monitor the performance of data access applications.

Performance Monitoring Infrastructure (PMI) method calls that are supported in the two existing Connection Managers (JDBC and J2C) are supported in this version of WebSphere Application Server. The calls include:

- ManagedConnectionsCreated
- ManagedConnectionsAllocated
- ManagedConnectionFreed
- ManagedConnectionDestroyed
- BeginWaitForConnection
- EndWaitForConnection
- ConnectionFaults
- Average number of ManagedConnections in the pool
- Percentage of the time that the connection pool is using the maximum number of ManagedConnections
- Average number of threads waiting for a ManagedConnection
- Average percent of the pool that is in use
- Average time spent waiting on a request
- Number of ManagedConnections that are in use
- Number of Connection Handles
- FreePoolSize
- UseTime

Java Specification Request (JSR) 77 requires statistical data to be accessed through managed beans (Mbeans) to facilitate this. The Connection Manager passes the ObjectNames of the Mbeans created for this pool. In the case of Java Message Service (JMS) *null* is passed in. The interface used is:

```
PmiFactory.createJ2CPerf(
    String pmiName, // a unique Identifier for JCA /JDBC.  This is the
                    // ConnectionFactory name.

    ObjectName providerName,// the ObjectName of the J2CResourceAdapter
                            // or JDBCProvider Mbean

    ObjectName factoryName // the ObjectName of the J2CConnectionFactory
                           // or DataSourceMbean.
)
```

The following Unified Modeling Language (UML) diagram shows how JSR 77 requires statistics to be reported:

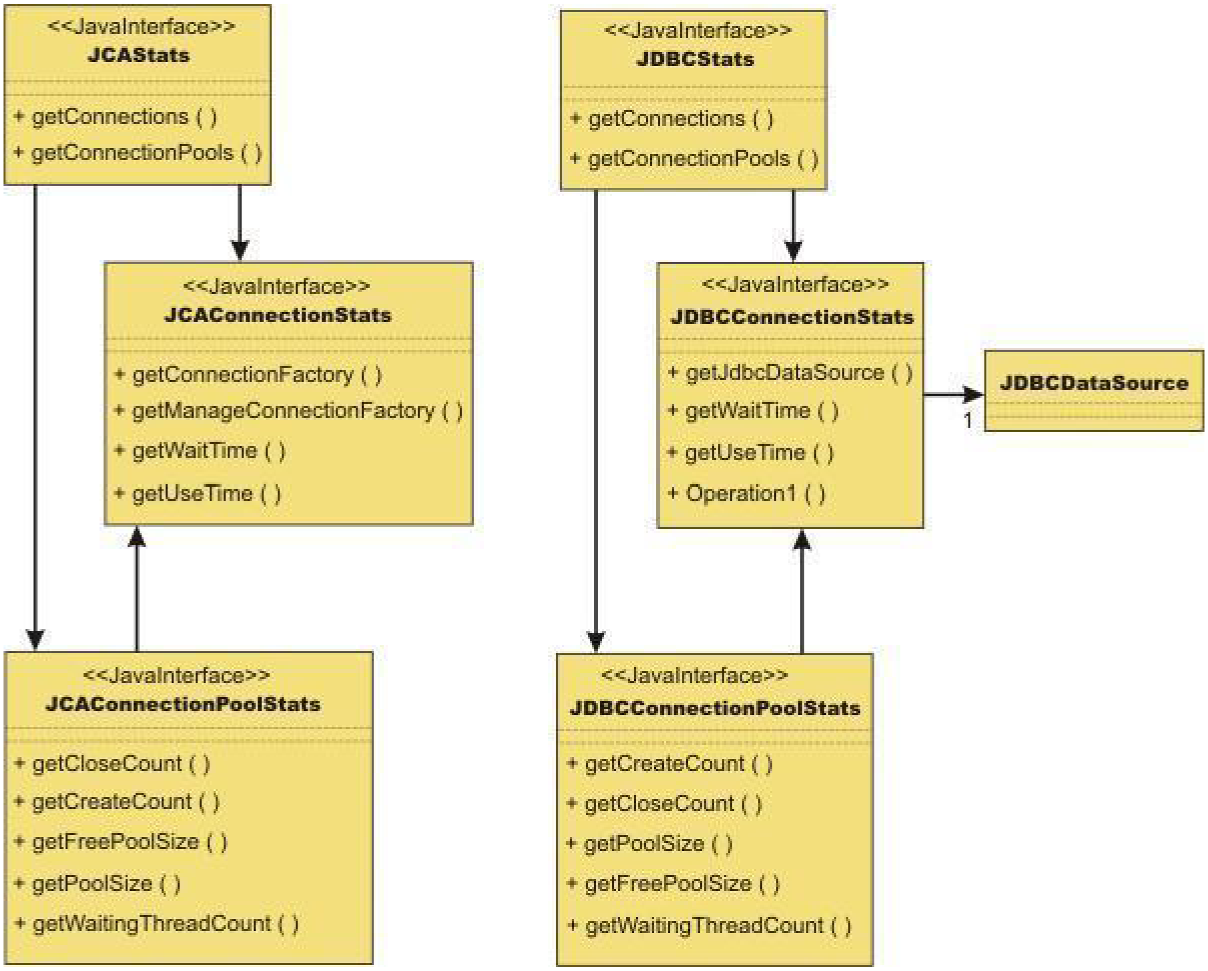


Figure 10.

JCAConnectionPoolStats and JDBCConnectionPoolStats objects do not have a direct implementing Mbean; the statistics are gathered through a call to PMI. A J2C resource adapter, and JDBC provider each contain a list of ConnectionFactory or DataSource ObjectNames, respectively. The ObjectNames are used by PMI to find the appropriate connection pool in the list of PMI modules.

The JCA 1.5 Specification allows an exception from the matchManagedConnection() method that indicates that the resource adapter requests that the connection not be pooled. In that case, statistics for that connection are provided separately from the statistics for the connection pool.

Connection life cycle

A ManagedConnection object is always in one of three states: *DoesNotExist*, *InFreePool*, or *InUse*.

Before a connection is created, it must be in the DoesNotExist state. After a connection is created, it can be in either the InUse or the InFreePool state, depending on whether it is allocated to an application.

Between these three states are *transitions*. These transitions are controlled by *guarding conditions*. A guarding condition is one in which *true* indicates when you can take the transition into another legal state. For example, you can make the transition from the InFreePool state to InUse state only if:

- the application has called the data source or connection factory getConnection() method (the *getConnection* condition)
- a free connection is available in the pool with matching properties (the *freeConnectionAvailable* condition)
- and one of the two following conditions are true:

– the getConnection() request is on behalf of a resource reference that is marked unsharable
– the getConnection() request is on behalf of a resource reference that is marked shareable but no shareable connection in use has the same properties.

This transition description follows:

```
InFreePool > InUse:
getConnection AND
freeConnectionAvailable AND
NOT(shareableConnectionAvailable)
```

Here is a list of guarding conditions and descriptions.

Table 18. Guarding conditions. Here is a list of guarding conditions and descriptions.

Condition	Description
ageTimeoutExpired	Connection is older then its ageTimeout value.
close	Application calls close method on the Connection object.
fatalErrorNotification	A connection has just experienced a fatal error.
freeConnectionAvailable	A connection with matching properties is available in the free pool.
getConnection	Application calls getConnection method on a data source or connection factory object.
markedStale	Connection is marked as stale, typically in response to a fatal error notification.
noOtherReferences	There is only one connection handle to the managed connection, and the Transaction Service is not holding a reference to the managed connection.
noTx	No transaction is in force.
poolSizeGTMin	Connection pool size is greater than the minimum pool size (minimum number of connections)
poolSizeLTMax	Pool size is less than the maximum pool size (maximum number of connections)
shareableConnectionAvailable	The getConnection() request is for a shareable connection, and one with matching properties is in use and available to share.
TxEnds	The transaction has ended.
unshareableConnectionRequest	The getConnection() request is for an unshareable connection.
unusedTimeoutExpired	Connection is in the free pool and not in use past its unused timeout value.

Getting connections

The first set of transitions covered are those in which the application requests a connection from either a data source or a connection factory. In some of these scenarios, a new connection to the database results. In others, the connection might be retrieved from the connection pool or shared with another request for a connection.

Every connection begins its life cycle in the DoesNotExist state. When an application server starts, the connection pool does not exist. Therefore, there are no connections. The first connection is not created until an application requests its first connection. Additional connections are created as needed, according to the guarding condition.

```
getConnection AND
NOT(freeConnectionAvailable) AND
poolSizeLTMax AND
(NOT(shareableConnectionAvailable) OR
unshareableConnectionRequest)
```

This transition specifies that a connection object is not created unless the following conditions occur:

- The application calls the getConnection() method on the data source or connection factory
- No connections are available in the free pool (NOT(freeConnectionAvailable))
- The pool size is less than the maximum pool size (poolSizeLTMax)
- If the request is for a sharable connection and there is no sharable connection already in use with the same sharing properties (NOT(shareableConnectionAvailable)) OR the request is for an unsharable connection (unshareableConnectionRequest)

All connections begin in the DoesNotExist state and are only created when the application requests a connection. The pool grows from 0 to the maximum number of connections as applications request new connections. The pool is **not** created with the minimum number of connections when the server starts.

If the request is for a sharable connection and a connection with the same sharing properties is already in use by the application, the connection is shared by two or more requests for a connection. In this case, a new connection is not created. For users of the JDBC API these sharing properties are most often *userid/password* and *transaction context*; for users of the Resource Adapter Common Client Interface (CCI) they are typically *ConnectionSpec*, *Subject*, and *transaction context*.

The transition from the InFreePool state to the InUse state is the most common transition when the application requests a connection from the pool.

```
InFreePool>InUse:
getConnection AND
freeConnectionAvailable AND
(unshareableConnectionRequest OR
NOT(shareableConnectionAvailable))
```

This transition states that a connection is placed in use from the free pool if:

- the application has issued a getConnection() call
- a connection is available for use in the connection pool (freeConnectionAvailable),
- and one of the following is true:
  - the request is for an unsharable connection (unsharableConnectionRequest)
  - no connection with the same sharing properties is already in use in the transaction. (NOT(sharableConnectionAvailable)).

Any connection request that a connection from the free pool can fulfill does not result in a new connection to the database. Therefore, if there is never more than one connection used at a time from the pool by any number of applications, the pool never grows beyond a size of one. This number can be less than the minimum number of connections specified for the pool. One way that a pool grows to the minimum number of connections is if the application has multiple concurrent requests for connections that must result in a newly created connection.

The idea of connection sharing is seen in the transition on the InUse state.

```
InUse>InUse:
getConnection AND
ShareableConnectionAvailable
```

This transition indicates that if an application requests a shareable connection (getConnection) with the **same** sharing properties as a connection that is already in use (ShareableConnectionAvailable), the existing connection is shared.

The same user (*user name* and *password*, or *subject*, depending on authentication choice) can share connections but only within the same transaction and only when all of the sharing properties match. For JDBC connections, these properties include the *isolation level*, which is configurable on the resource-reference (IBM WebSphere extension) to data source default. For a resource adapter factory connection, these properties include those specified on the ConnectionSpec object. Because a transaction is normally associated with a single thread, you should **never** share connections across threads.

Note: It is possible to see the same connection on multiple threads at the same time, but this situation is an error state usually caused by an application programming error.

Returning connections

All of the transitions discussed previously involve getting a connection for application use. With that goal, the transitions result in a connection closing, and either returning to the free pool or being destroyed. Applications should explicitly close connections (note: the connection that the user gets back is really a connection handle) by calling close() on the connection object. In most cases, this action results in the following transition:

```
InUse>InFreePool:
(close AND
noOtherReferences AND
NoTx AND
UnshareableConnection)
OR
(ShareableConnection AND
TxEnds)
```

Conditions that cause the transition from the InUse state are:

- If the application or the container calls close() (producing the close condition) and there are no references (the noOtherReferences condition) either by the application (in the application sharing condition) or by the transaction manager (in the NoTx condition, meaning that the transaction manager holds a reference when the connection is enlisted in a transaction), the connection object returns to the free pool.
- If the connection was enlisted in a transaction but the transaction manager ends the transaction (the txEnds condition), and the connection was a shareable connection (the ShareableConnection condition), the connection closes and returns to the pool.

When the application calls close() on a connection, it is returning the connection to the pool of free connections; it is **not** closing the connection to the data store. When the application calls close() on a currently shared connection, the connection is *not returned* to the free pool. Only after the application drops the last reference to the connection, and the transaction is over, is the connection returned to the pool. Applications using unsharable connections must take care to close connections in a timely manner. Failure to do so can starve out the connection pool, making it impossible for any application running on the server to get a connection.

When the application calls close() on a connection enlisted in a transaction, the connection is not returned to the free pool. Because the transaction manager must also hold a reference to the connection object, the connection cannot return to the free pool until the transaction ends. Once a connection is enlisted in a transaction, you cannot use it in any other transaction by any other application until after the transaction is complete.

There is a case where an application calling close() can result in the connection to the data store closing and bypassing the connection return to the pool. This situation happens if one of the connections in the pool is considered stale. A connection is considered stale if you can no longer use it to contact the data store. For example, a connection is marked stale if the data store server is shut down. When a connection is marked as stale, the entire pool is cleaned out by default because it is very likely that all of the connections are stale for the same reason (or you can set your configuration to clean just the failing connection). This cleansing includes marking all of the currently InUse connections as stale so they are destroyed upon closing. The following transition states the behavior on a call to close() when the connection is marked as stale:

```
InUse>DoesNotExist:
close AND
markedStale AND
NoTx AND
noOtherReferences
```

This transition states that if the application calls close() on the connection and the connection is marked as stale during the pool cleansing step (markedStale), the connection object closes to the data store and is not returned to the pool.

Finally, you can close connections to the data store and remove them from the pool.

This transition states that there are three cases in which a connection is removed from the free pool and destroyed.

1. If a fatal error notification is received from the resource adapter (or data source). A fatal error notification (FatalErrorNotification) is received from the resource adaptor when something happens to the connection to make it unusable. All connections currently in the free pool are destroyed.
2. If the connection is in the free pool for longer than the unused timeout period (UnusedTimeoutExpired) and the pool size is greater than the minimum number of connections (poolSizeGTMin), the connection is removed from the free pool and destroyed. This mechanism enables the pool to shrink back to its minimum size when the demand for connections decreases.
3. If an age timeout is configured and a given connection is older than the timeout. This mechanism provides a way to recycle connections based on age.

Unshareable and shareable connections

The application server supports both unshareable and shareable connections. An unshareable connection is not shared with other components in the application. The component using this connection has full control of this connection.

Access to a resource marked as unshareable means that there is a one-to-one relationship between the connection handle a component is using and the physical connection with which the handle is associated. This access implies that every call to the getConnection method returns a connection handle solely for the requesting user. Typically, you must choose unshareable if you might do things to the connection that could result in unexpected behavior occurring in another application that is sharing the connection (for example, unexpectedly changing the isolation level).

Marking a resource as shareable allows for greater scalability. Instead of retrieving a new physical connection from the connection pool for every getConnection() invocation, the physical connection (that is, managed connection) is shared through multiple connection handles, as long as each getConnection request has the same connection properties. However, sharing a connection means that each user must not do anything to the connection that could change its behavior and disrupt a sharing partner (for example, changing the isolation level). The user also cannot code an application that assumes sharing to take place because it is up to the run time to decide whether or not to share a particular connection.

Connection property requirements

To permit sharing of connections used within the same transaction, the following data source properties must be the same:

- Java Naming and Directory Interface (JNDI) name. While not actually a connection property, this requirement simply means that you can only share connections from the same data source in the same server.
- Resource authentication
- In relational databases:
  - Isolation level (corresponds to access intent policies applied to CMP beans)

- Readonly
- Catalog
- TypeMap

For more information on sharing a connection with a CMP bean, see the topic Sharing a connection with a CMP bean.

To permit sharing of connections within the same transaction, the following properties must be the same for the connection factories:

- JNDI name. While not actually a connection property, this requirement simply means that you can only share connections from the same connection factory in the same server.
- Resource authentication

In addition, the ConnectionSpec object that is used to get the connection must also be the same.

Note: Java Message Service (JMS) connections cannot be shared with non-JMS connections.

JMS connections for the WebSphere MQ JMS Provider cannot be shareable because they are non-transactional, and the Java™ EE Connector Architecture (JCA) specification only allows transactional resources to be shareable. If the res-sharing-scope is set to shareable in a JMS resource reference, the setting will be ignored and unshareable connections will be used. However, JMS sessions for MQ are transactional, and can be shareable. JMS sessions are shareable by default, and APAR PK59605 provides the ability to specify unshareable sessions.

JMS connections for the Default Messaging Provider are different. With the Default Messaging Provider, connections can be shareable. Sessions, on the other hand, are not managed by a connection pool, and therefore cannot be shareable or unshareable.

Sharing a connection with a CMP bean

The application server allows you to share a physical connection among a CMP bean, a BMP bean, and a JDBC application to reduce the resource allocation or deadlock scenarios. There are several ways to ensure that all of these entity beans and the JDBC applications are sharing the same physical connection.

- **Sharing a connection between CMP beans or methods**

  When all CMP bean methods use the same access intent, they all share the same physical connection. A different access intent policy triggers the allocation of a different physical connection. For example, a CMP bean has two methods; method 1 is associated with `wsPessimisticUpdate` intent, whereas method 2 has `wsOptimisticUpdate` access intent. Method 1 and method 2 cannot share the same physical connection within a transaction. In other words, an XA data source is required to run in a global transaction.

  You can experience some deadlocks from a database if both methods try to access the same table. Therefore, sharing a connection is determined by the access intents that are defined in the CMP methods.

- **Sharing a connection between CMP and BMP beans**

  Remember to first verify that the **getConnection** methods of both the BMP bean and the CMP bean set the same connection properties. To match the authentication type of the CMP bean resource, set the authentication type of the BMP bean resource to container-managed, which is designated in the deployment descriptor as `res-auth = Container`.

  Additionally, use one of the following options to ensure connection-sharing between the bean types:

  - Define the same access intent on both CMP and BMP bean methods. Because both use the same access intent, they share the same physical connection. The advantage to using this option is that the backend is transparent to a BMP bean. However, this option also makes the BMP non-portable because it uses the WebSphere extended API to handle the isolation level. For more information, refer to the code example in the topic, Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans.

  - Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level that is specified on the resource reference to look up a data source and a connection. This option is more of a manual process, and the isolation level might be different from database to database. For more information refer to the isolation level and access intent mapping table in the topic, Access intent isolation levels and update locks, and the topic, Isolation level and resource reference section.
- **Sharing a connection between CMP and a JDBC application that is used by a servlet or a session bean**Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level specified on the resource reference to look up a data source and a connection. For more information see the topic, Access intent isolation levels, and the topic, Isolation level and resource reference section.

Factors that determine sharing

The listing here is not an exhaustive one. The product might or might not share connections under different circumstances.

- Only connections acquired with the same resource reference (resource-ref) that specifies the res-sharing-scope as shareable are candidates for sharing. The resource reference properties of res-sharing-scope and res-auth and the IBM extension isolationLevel help determine if it is possible to share a connection. IBM extension isolationLevel is stored in IBM deployment descriptor extension file; for example: ibm-ejb-jar-ext.xmi.

  Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

  - For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
  - For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

  However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

  The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.
- You can only share connections that are requested with the same properties.
- Connection sharing only occurs between different component instances if they are within a transaction (container- or user-initiated transaction).
- Connection sharing only occurs within a sharing boundary. Current® sharing boundaries include *Transactions* and *LocalTransactionContainment* (LTC) boundaries.
- Connection sharing rules within an LTC Scope:
  - For shareable connections, only *Connection Reuse* is allowed within a single component instance. Connection reuse occurs when the following actions are taken with a connection: get, use, commit/rollback, close; get, use, commit/rollback, close. Note that if you use the LTC resolution-control of *ContainerAtBoundary* then no start/commit is needed because that action is handled by the container.

    The connection returned on the second *get* is the same connection as that returned on the first *get* (if the same properties are used). Because the connection use is serial, only one connection handle to the underlying physical connection is used at a time, so true connection sharing does not take place. The term *"reuse"* is more accurate.

  More importantly, the *LocalTransactionContainment* boundary enclosing both *get* actions is not complete; no cleanUp() method is invoked on the ManagedConnection object. Therefore the second *get* action inherits all of the connection properties set during the first getConnection() call.

- Shareable connections between transactions (either container-managed transactions (CMT), bean-managed transactions (BMT), or LTC transactions) follow these caching rules:
  - In general, setting properties on shareable connections is not allowed because a user of one connection handle might not anticipate a change made by another connection handle. This limitation is part of the Java Platform, Enterprise Edition (Java EE) standard.
  - General users of resource adapters can set the connection properties on the connection factory getConnection() call by passing them in a ConnectionSpec.

    However, the properties set on the connection during one transaction are not guaranteed to be the same when used in the next transaction. Because it is not valid to share connections outside of a sharing scope, connection handles are moved off of the physical connection with which they are currently associated when a transaction ends. That physical connection is returned to the free connection pool. Connections are cleaned before going in the free pool. The next time the handle is used, it is automatically associated with an appropriate connection. The appropriateness is based on the security login information, connection properties, and (for the JDBC API) the isolation level specified in the extended resource reference, passed in on the original request that returned the current handle. Any properties set on the connection after it was retrieved are lost.
  - For JDBC users, the application server provides an extension to enable passing the connection properties through the ConnectionSpec.

    Use caution when setting properties and sharing connections in a local transaction scope. Ensure that other components with which the connection is shared are expecting the behavior resulting from your settings.
- You cannot set the isolation level on a shareable connection for the JDBC API using a relational resource adapter in a global transaction. The product provides an extension to the resource reference to enable you to specify the isolation level. If your application requires the use of multiple isolation levels, create multiple resource references and map them to the same data source or connection factory.

Maximal connection sharing

To maximize connection sharing opportunities for an application, ensure that each component has the local transaction containment (LTC) Resolver attribute set to **ContainerAtBoundary**. This setting specifies that the component container, rather than the application code, resolves all resource manager local transactions (RMLTs) within the LTC scope. The container begins an RMLT when a connection is first used within the LTC scope, and completes it automatically at the end of the LTC scope.

See the topic, Configuring transactional deployment attributes, for instructions on setting the transaction resolution control and other attributes.

Connection sharing violations

There is a new exception, the sharing violation exception, that the resource adapter can issue whenever an operation violates sharing requirements. Possible violations include changing connection attributes, security settings, or isolation levels, among others. When such a mutable operation is performed against a managed connection, the sharing violation exception can occur when both of the following conditions are true:
- The number of connection handles associated with the managed connection is more than one.
- The managed connection is associated with a transaction, either local or XA.

Both the component and the J2C run time might need to detect this sharing violation exception, depending on when and how the managed connection becomes unshareable. If the managed connection becomes unshareable because of an operation through the connection handle (for example, you change the isolation level), then the component needs to process the exception. If the managed connection becomes unshareable without being recognized by the application server (due to some component interaction with the connection handle), then the resource adapter can reject the creation of a connection handle by issuing the sharing violation exception.

Connection handles

A connection handle is a representation of a physical connection. To use a backend resource, such as a relational database in WebSphere Application Server, you must get a connection to that resource. When you call the getConnection() method, you get a connection handle returned. The handle is not the physical connection. The physical connection is managed by the connection manager.

There are two significant configurations that affect how connection handles are used and how they behave. The first is the res-sharing-scope, which is defined by the resource-reference used to look up the DataSource or Connection Factory. This property tells the connection manager whether or not you can share this connection.

The second factor that affects connection handle behavior is the usage pattern. There are essentially two usage patterns. The first is called the get/use/close pattern. It is used within a single method and without calling another method that might get a connection from the same data source or connection factory. An application using this pattern does the following:
1. gets a connection
2. does its work
3. commits (if appropriate)
4. closes the connection.

The second usage pattern is called the cached handle pattern. This is where an application:
1. gets a connection
2. begins a global transaction
3. does work on the connection
4. commits a global transaction
5. does work on the connection again

A cached handle is a connection handle that is held across transaction and method boundaries by an application. Keep in mind the following considerations for using cached handles:
- Cached handle support requires some additional connection handle management across these boundaries, which can impact performance. For example, in a JDBC application, Statements, PreparedStatements, and ResultSets are closed implicitly after a transaction ends, but the connection remains valid.
- You are encouraged not to cache the connection across the transaction boundary for shareable connections; the get/use/close pattern is preferred.
- Caching of connection handles across servlet methods is limited to JDBC and Java Message Service (JMS) resources. Other non-relational resources, such as Customer Information Control System (CICS) or IMS objects, currently cannot have their connection handles cached in a servlet; you need to get, use, and close the connection handle within each method invocation. (This limitation only applies to single-threaded servlets because multithreaded servlets do not allow caching of connection handles.)
- You cannot pass a cached connection handle from one instance of a data access client to another client instance. Transferring between client instances creates the problematic contingency of one instance using a connection handle that is referenced by another. This relationship can only cause problems because connection handle management code processes tasks for each client instance separately. Hence, connection handle transfers result in run-time scenarios that trigger exceptions. For example:
  1. The application code of a client instance that receives a transferred handle closes the handle.
  2. If the client instance that retains the original reference to the handle tries to reclaim it, the application server issues an exception.

The following code segment shows the cached connection pattern.

```
Connection conn = ds.getConnection();
ut.begin();
conn.prepareStatement("....."); --> Connection runs in global transaction mode
...
ut.commit();
conn.prepareStatement(".....");   ---> Connection still valid but runs in autoCommit(True);
...
```

Unshareable connections

Some characteristics of connection handles retrieved with a res-sharing-scope of unshareable are described in the following sections.

- The possible benefits of unshared connections
  - Your application always maintains a direct link with a physical connection (managed connection).
  - The connection always has a one-to-one relationship between the connection handle and the managed connection.
  - In most cases, the connection does not close until the application closes it.
  - You can use a cached unshared connection handle across multiple transactions.
  - The connection can have a performance advantage in some cached handle situations. Because unshared connections do not have the overhead of moving connection handles off managed connections at the end of the transaction, there is less overhead in using a cached unshared connection.
- The possible drawbacks of unshared connections
  - Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection (with the same properties) using the same data source or connection factory (same resource-ref) then you use multiple physical connections when you use unshareable connections.
  - Wasted connections. It is important not to keep the connection handle open (that is, your application does not call the close() method) any longer then it is needed. As long as an unshareable connection is open, the physical connection is unavailable to any other component, even if your application is not currently using that connection. Unlike a shareable connection, an ushareable connection is not closed at the end of a transaction or servlet call.
  - Deadlock considerations. Depending on how your components interact with the database within a transaction, using unshared connections can lead to deadlock in the database. For example, within a transaction, component A gets a connection to data source X and updates table 1, and then calls component B. Component B gets another connection to data source X, and updates/reads table 1 (or even worse the same row as component A). In some circumstances, depending on the particular database, its locking scheme, and the transaction isolation level, a deadlock can occur.

    In the same scenario, but with a shared connection, deadlock does not occur because all the work is done on the same connection. It is worth noting that when writing code that uses shared connections, you use a strategy that calls for multiple work items to be performed on the same connection, possibly within the same transaction. If you decide to use an unshareable connection, you must set the maximum connections property on the connection factory or data source correctly. An exception might occur for waiting connection requests if you exceed the maximum connections value, and unshareable connections are not being closed before the connection wait time-out is exceeded.

Shareable connections

Some characteristics of connection handles that are retrieved with a res-sharing-scope of shareable are described in the following sections.

- The possible benefits of shared connections
  - Within an instance of connection sharing, application components can share a managed connection with one or more connection handles, depending on how the handle is retrieved and which connection properties are used.

- They can more efficiently use resources. Shareable connections are not valid outside of their sharing boundary. For this reason, at the end of a sharing boundary (such as a transaction) the connection handle is no longer associated with the managed connection it was using within the sharing boundary (this applies only when using the cached handle pattern). The managed connection is returned to the free connection pool for reuse. Connection resources are not held longer than the end of the current sharing scope.

  If the cached handle pattern is used, then the next time the handle is used within a new sharing scope, the application server run time ensures that the handle is reassociated with a managed connection that is appropriate for the current sharing scope, and has the same properties with which the handle was originally retrieved. Remember that it is not appropriate to change properties on a shareable connection. If properties are changed, other components that share the same connection might experience unexpected behavior. Futhermore, when using cached handles, the value of the changed property might not be remembered across sharing scopes.

- The possible drawbacks of shared connections
  - Sharing within a single component (such as an enterprise bean and its related Java objects) is not always supported. The current specification allows resource adapters the choice of only allowing one active connection handle at a time.

    If a resource adapter chooses to implement this option then the following scenario results in an invalid handle exception: A component using shareable connections gets a connection and uses it. Without closing the connection, the component calls a utility class (Java object) that gets a connection handle to the same managed connection and uses it. Because the resource adapter only supports one active handle, the first connection handle is no longer valid. If the utility object returns without closing its handle, the first handle is not valid and triggers an exception at any attempt to use it.

    Note: This exception occurs only when calling a utility object (a Java object).

    Not all resource adapters have this limitation; it occurs only in certain implementations. The WebSphere Relational Resource Adapter (RRA) does not have this limitation. Any data source used through the RRA does not have this limitation. If you encounter a resource adapter with this limitation you can work around it by serializing your access to the managed connection. If you always close your connection handle before getting another (or close your handle before calling code that gets another handle), and before returning from a method, you can allow two pieces of code to share the same managed connection. You simply cannot use the connection for both events at the same time.
  - Trying to change the isolation level on a shareable JDBC-based connection in a global transaction (that is supported by the RRA) causes an exception. The correct way to get connections with different transaction isolation levels is by configuring the IBM extended resource-reference.
  - Closing connection handles for shareable connections by an application is NOT supported and causes errors. However, you can avoid this limitation by using the Relational Resource Adapter.

Lazy connection association optimization

The Java Platform, Enterprise Edition (Java EE) Connector (J2C) connection manager implemented smart handle support. This technology enables allocation of a connection handle to an application while the managed connection associated with that connection handle is used by other applications (assuming that the connection is not being used by the original application). This concept is part of the Java EE Connector Architecture (JCA) 1.5 specification. (You can find it in the JCA 1.5 specification document in the section entitled “Lazy Connection Association Optimization.”) Smart handle support introduces use a method on the ConnectionManager object, the LazyAssociatableConnectionManager() method , and a new marker interface, the DissociatableManagedConnection class. You must configure the provider of the resource adapter to make this functionality available in your environment. (In the case of the RRA, WebSphere Application Server itself is the provider.) The following code snippet shows how to include smart handle support:

```
package javax.resource.spi;
import javax.resource.ResourceException;
```

```
interface LazyAssociatableConnectionManager { // application server
    void associateConnection(
        Object connection, ManagedConnectionFactory mcf,
        ConnectionRequestInfo info) throws ResourceException;
}

interface DissociatableManagedConnection { // resource adapter
    void dissociateConnections() throws ResourceException;
}
```

This DissociatableManagedConnection interface introduces another state to the Connection object: inactive. A Connection can now be active, closed, and inactive. The connection object enters the inactive state when a corresponding ManagedConnection object is cleaned up. The connection stays inactive until an application component attempts to re-use it. Then the resource adapter calls back to the connection manager to re-associate the connection with an active ManagedConnection object.

Transaction type and connection behavior

All connection usage occurs within the scope of either a global transaction or a local transaction containment (LTC) boundary. Each transaction type places different requirements on connections and impacts connection settings differently.

Connection sharing and reuse

You can share connections within a global transaction scope (assuming other sharing rules are met). You can also share connections within a shareable LTC. You can *serially reuse* connections within an LTC scope. A get/use/close connection pattern followed by another instance of get/use/close (to the same data source or connection factory) enables you to reuse the same connection. See the topic, Unshareable and shareable connections for more details.

JDBC AutoCommit behavior

All JDBC connections, when first obtained through a getConnection() call, contain the setting AutoCommit = TRUE by default. However, different transaction scope and settings can result in changing, or simply overriding, the AutoCommit value.

- If you operate within an LTC and have its resolution-control set to *Application*, AutoCommit remains *TRUE* unless changed by the application.
- If you operate within an LTC and have its resolution-control set to *ContainerAtBoundary*, the application must **not** touch the AutoCommit setting. TheWebSphere Application Server run time sets the AutoCommit value to *FALSE* before work begins, then commits or rolls back the work, as appropriate, at the end of the LTC scope.
- If you use a connection within a global transaction, the database ignores the AutoCommit setting so that the transaction service that controls the commit and rollback processing can manage the transaction. This action takes place upon first use of the connection to do work, regardless of the user changing the AutoCommit setting. After the transaction completes, the AutoCommit value returns to the value it had before the first use of the connection. So even if the AutoCommit value is set to *TRUE* before the connection is used in a global transaction, you need not set the value to *FALSE* because the value is ignored by the database. In this example, after the transaction completes, the AutoCommit value of the connection returns to *TRUE*.
- If you use multiple distinct connections within a global transaction, all work is guaranteed to commit or roll back together. This is not the case for a local transaction containment (LTC scope). Within an LTC, work done on one connection commits or rolls back independently from work done on any other connection within the LTC.

One-phase commit and two-phase commit connections

The type and number of resource managers, such as a database server, that must be accessed by an application often determines the application transaction requirements. Consequently each type of resource manager places different requirements on connection behavior.

- A two-phase commit resource manager can support two-phase coordination of a transaction. That support is necessary for transactions that involve other resource managers; these transactions are global transactions. See the topic, Transaction support in WebSphere Application Server for further explanation.
- A one-phase commit resource manager supports only one-phase transactions, or LTC transactions, in which that resource is the sole participating datastore. See the topic, Transaction support in WebSphere Application Server for further explanation.

One-phase commit resources are such that work being done on a one phase connection cannot mix with other connections and ensure that the work done on all of the connections completes or fails atomically. The product does not allow more than one one-phase commit connection in a global transaction. Futhermore, it does not allow a one-phase commit connection in a global transaction with one or more two-phase commit connections. You can coordinate only multiple two-phase commit connections within a global transaction.

WebSphere Application Server provides *last participant support*, which enables a single one-phase commit resource to participate in a global transaction with one or more two-phase commit resources.

Note that any time that you do multiple getConnection() calls using a resource reference that specifies res-sharing-scope=Unshareable, you get multiple physical connections. This situation also occurs when res-sharing-scope=Shareable, but the sharing rules are broken. In either case, if you run in a global transaction, ensure the resources involved are enabled for two-phase commit (also sometimes referred to as *JTA Enabled*). Failure to do so results in an XA exception that logs the following message:

```
WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable
resources has occurred.
```

Application scoped resources

Use this page to view brief descriptions of the resources that are bundled with your application. You can view individual resource settings by clicking the resource name.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name*** > **Application scoped resources**.

The **Application scoped resources** link is not present if there are no application scoped resources.

Each table row corresponds to a resource that is bundled with your application. Click a resource name or the corresponding provider name to view an administrative console page where you can edit the object configuration settings.

Name:

Specifies the administrative name that was assigned to this resource.

Click this name to view a page where you can edit the configuration settings.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of the resource.

Information	Value
Data type	String

Resource type:

Specifies the type of resource, such as a data source or a J2C connection factory.

Provider:

Specifies the resource provider that supplies the class information for this resource object.

Click the provider name to view a page where you can edit the configuration settings.

Description:

Specifies a text description of the resource.

Data access: Resources for learning

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

These links are provided for convenience. Often, the information is not specific to this product, but it can be useful for understanding concepts or functions used by the application server. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information:
- “Technologies for data access”
- “Databases” on page 108
- “Tools” on page 108

Technologies for data access

- JDBC 3.0 API Documentation
- **Java Persistence API:**
  - Java Persistence API FAQ
  - Introduction to Spring 2 and JPA
- J2EE Connector Architecture Version 1.5 specification
- Enterprise JavaBeans Technology (Source for download of the Enterprise Javabeans 3.0 specification)
- Java 2 Platform, Enterprise Edition (J2EE)
- **Container-managed relationships:** Enterprise JavaBeans 2.0 Container-Managed Persistence Example. Although this article addresses the EJB 2.0 specification, you might find parts of it pertinent to your environment.
- **Resource adapters:** The J2EE Connector Architecture Resource AdapterDeveloper Technical Articles & Tips -- Articles: Database Access (Sun Developer Network)
- Java Management Extensions (JMX)
- **Miscellaneous articles from the Sun Developer Network and IBM developerWorks websites:**
  - Sharing connections in WebSphere Application Server V5 This article is still pertinent to WebSphere Application Server Version 6.0 and later. However, be aware that the container-managed authentication type is deprecated.
  - Database authentication in WebSphere Application Server V5 This article is still pertinent to WebSphere Application Server Version 6.0 and later. However, be aware that the container-managed authentication type is now deprecated.

  - Understanding WebSphere Application Server EJB access intents
- Supported hardware, software, and APIs

Databases

- **Cloudscape:**
  - IBM Cloudscape product information
  - IBM Cloudscape information center
- DB2 database software
- Informix®

Tools

- Rational Application Developer for WebSphere Software

Service Data Objects: Resources for learning

Use the following links to find relevant supplemental information about the service data object and various other functions that can be used with it. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

Service Data Objects

For an introduction to Service Data Objects, refer to:
- Introduction to Service Data Objects

For an overview of the Service Data Objects specification, refer to:
- Service Data Objects

A good place to start to learn about the Eclipse Modeling Framework is:
- EMF Eclipse Modeling Framework

Information about XSD to SDO/EMF mapping for Version 6 can be found at:
- Authoring XML Schemas for use with EMF

Web application presentation layer technologies

For a brief overview of JavaServer Faces, refer to:
- Java Sun J2EE 1.4 tutorial

Good places to start to learn about JavaServer Pages Standard Tag Library are:
- JavaServer Pages Standard Tag Library
- A JSTL primer, Part 1: The expression language

Java Persistence API (JPA) architecture

Data persistence is the ability to maintain data between application executions. Persistence is vital to enterprise applications because of the required access to relational databases. Applications that are developed for this environment must manage persistence themselves or use third-party solutions to handle database updates and retrievals with persistence. The Java Persistence API (JPA) provides a mechanism for managing persistence and object-relational mapping and functions for the EJB 3.0 and EJB 3.1 specifications.

The JPA specification defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations. JPA is based on the Java programming model that applies to Java EE environments, but JPA can function within a Java SE environment for testing application functions.

JPA represents a simplification of the persistence programming model. The JPA specification explicitly defines the object-relational mapping, rather than relying on vendor-specific mapping implementations. JPA standardizes the important task of object-relational mapping by using annotations or XML to map objects into one or more tables of a database. To further simplify the persistence programming model:

- The EntityManager API can persist, update, retrieve, or remove objects from a database
- The EntityManager API and object-relational mapping metadata handle most of the database operations without requiring you to write JDBC or SQL code to maintain persistence
- JPA provides a query language, extending the independent EJB querying language (also known as JPQL), that you can use to retrieve objects without writing SQL queries specific to the database you are working with.

JPA is designed to operate both inside and outside of a Java Enterprise Edition (Java EE) container. When you run JPA inside a container, the applications can use the container to manage the persistence context. If there is no container to manage JPA, the application must handle the persistence context management itself. Applications that are designed for container-managed persistence do not require as much code implementation to handle persistence, but these applications cannot be used outside of a container. Applications that manage their own persistence can function in a container environment or a Java SE environment.

Java EE containers that support the EJB 3.x programming model must support a JPA implementation, also called a persistence provider. A JPA persistence provider uses the following elements to allow for easier persistence management in an EJB 3.x environment:

Persistence unit

Consists of the declarative metadata that describes the relationship of entity class objects to a relational database. The EntityManagerFactory uses this data to create a persistence context that can be accessed through the EntityManager.

EntityManagerFactory

Used to create an EntityManager for database interactions. The application server containers typically supply this function, but the EntityManagerFactory is required if you are using JPA application-managed persistence. An instance of an EntityManagerFactory represents a Persistence Context.

Persistence context

Defines the set of active instances that the application is manipulating currently. The persistence context can be created manually or through injection.

EntityManager

The resource manager that maintains the active collection of entity objects that are being used by the application. The EntityManager handles the database interaction and metadata for object-relational mappings. An instance of an EntityManager represents a Persistence Context. An application in a container can obtain the EntityManager through injection into the application or by looking it up in the Java component name-space. If the application manages its persistence, the EntityManager is obtained from the EntityManagerFactory.

Attention: Injection of the EntityManager is only supported for the following artifacts:

- EJB 3.x session beans
- EJB 3.x message-driven beans
- Servlets, Servlet Filters, and Listeners
- JSP tag handlers which implement interfaces javax.servlet. jsp.tagext.Tag and javax.servlet.jsp.tagext.SimpleTag
- JavaServer Faces (JSF) managed beans
- the main class of the application client.

Entity objects
A simple Java class that represents a row in a database table in its simplest form. Entities objects can be concrete classes or abstract classes. They maintain states by using properties or fields.

For more information about persistence, see the section on Java Persistence API Architecture and the section on Persistence in the Apache OpenJPA User Guide. For more information and examples on specific elements of persistence, see the sections on the EntityManagerFactory, and the EntityManager in the Apache OpenJPA User Guide.

JPA for WebSphere Application Server

Java Persistence API (JPA) 2.0 for WebSphere Application Server is built on the Apache OpenJPA 2.x open source project.

Apache OpenJPA is a compliant implementation of the JPA specification. Using OpenJPA as a base implementation, WebSphere Application Server employs extensions to provide additional features and utilities for WebSphere Application Server customers. Because JPA for WebSphere Application Server is built from OpenJPA, all OpenJPA function, extensions, and configurations are unaffected by the WebSphere Application Server extensions. You do not need to make changes to OpenJPA applications to use these applications in WebSphere Application Server.

JPA for WebSphere Application Server provides more than compatibility with OpenJPA. JPA for WebSphere Application Server contains a set of tools for application development and deployment. Other features of JPA for WebSphere Application Server include support for DB2 Optim™ pureQuery Runtime, DB2 optimizations, JPA Access Intent, enhanced tracing capabilities, command scripts, and translated message files. The provider of JPA for WebSphere Application Server is `com.ibm.websphere.persistence.PersistenceProviderImpl`.

Apache OpenJPA supports the use of properties to configure the persistent environment. JPA for WebSphere Application Server properties can be specified with either the openjpa or wsjpa prefix. You can mix the openjpa and wsjpa prefixes as you wish for a common set of properties. Exceptions to the rule are wsjpa specific configuration properties, which use the wsjpa prefix. When a JPA for WebSphere Application Server-specific property is used with the openjpa prefix, a warning message is logged indicating that the offending property is treated as a wsjpa property. The reverse does not hold true for the openjpa prefix. In that case, the offending property is ignored.

wsjpaversion command

Use this command-line tool to find out information about the installed version of Java Persistence API (JPA) for WebSphere Application Server.

Run the JPA commands (.bat on Windows or .sh on UNIX) from the `<*profile_root*>/bin` directory, to make sure that you have the latest version of the commands for your release.

Syntax

The command syntax is as follows:

Usage

The version tool can be useful when debugging problems with JPA in applications and providing customer support teams with the information about the current JPA environment.

The command is run from the *<profile_root>* directory.

Examples

Find the version information of your JPA installation example output:

```
[root@atlanta bin]# ./wsjpaversion.sh
WSJPA 2.1.0-SNAPSHOT
version id: WSJPA-2.1.0-SNAPSHOT-r1119:2233
WebSphere JPA svn revision: 1119:2233

OpenJPA 2.1.0-SNAPSHOT
version id: openjpa-2.1.0-SNAPSHOT-r422266:1069208
Apache svn revision: 422266:1069208

os.name: Linux
os.version: 2.6.18-238.1.1.el5
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
 /root/tc/WASX/as/dev/JavaEE/j2ee.jar
 /root/tc/WASX/as/plugins/com.ibm.ws.jpa.jar
 /root/tc/WASX/as/plugins/com.ibm.ws.prereq.commons-collections.jar

/root/tc/WASX/as/profiles/AppSrv01/bin
[root@atlanta bin]#
```

On Windows operating systems, the output looks like the following:

```
D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin>wsjpaversion.bat
WSJPA 2.1.0-SNAPSHOT
version id: WSJPA-2.1.0-SNAPSHOT-r1119:2216
WebSphere JPA svn revision: 1119:2216
OpenJPA 2.1.0-SNAPSHOT
version id: openjpa-2.1.0-SNAPSHOT-r422266:1063829
Apache svn revision: 422266:1063829
os.name: Windows 7
os.version: 6.1
os.arch: amd64
java.version: 1.6.0
java.vendor: IBM Corporation
java.class.path:
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\dev\JavaEE\j2ee.jar
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\plugins\com.ibm.ws.jpa.jar
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\plugins\com.ibm.ws.prereq.
commons-collections.jar
        .
        C:\Program Files (x86)\IBM\Java60\jre\lib\ext\QTJava.zip
user.dir: D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin
D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin>
```

Examples

Find the version information of your JPA installation example output:

```
C:\was70-GM>profiles\a1002.07\bin\wsjpaversion.bat
WSJPA 2.0.0-SNAPSHOT
version id: WSJPA-2.0.0-SNAPSHOT-r1118:1843
```

```
revision: 1118:1843

OpenJPA 2.0.0-SNAPSHOT
version id: openjpa-2.0.0-SNAPSHOT-r422266:897308
Apache svn revision: 422266:897308

os.name: Windows XP
os.version: 5.1 build 2600 Service Pack 2
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
        C:\was70-GM\feature_packs\jpa\dev\JavaEE\j2ee.jar
        C:\was70-GM\feature_packs\jpa\plugins\com.ibm.ws.jpa.jar
        C:\was70-GM\plugins\com.ibm.ws.prereq.commons-collections.jar

user.dir: C:\was70-GM\plugins\com.ibm.ws.jpa.jar
```

wsjpa properties

The extension properties of Java Persistence API (JPA) for WebSphere Application Server can be specified with the openjpa or wsjpa prefix. This topic features the wsjpa properties.

wsjpa.AccessIntent

Use this property to define a TaskName that in the `persistence.xml` file using the wsjpa.AccessIntent property name in a persistence unit. The property value is a list of TaskNames, entity types and access intent definitions.

For more information and examples on how the wsjpa.AccessIntent property is used, see the topic Specifying TaskName in a JPA persistence unit.

wsjpa.jdbc.Schema

Specifies the schema name in a DB2 package collection when using multiple DB2 package collections.

For more information about using the wsjpa.jdbc.Schema property see the topic, Configuring pureQuery to use multiple DB2 package collections.

wsjpa.jdbc.CollectionId

Specifies the collection Id name in a DB2 package collection when using multiple DB2 package collections.

For more information about using the wsjpa.jdbc.CollectionId property see the topics, Configuring pureQuery to use multiple DB2 package collections and Configuring data source JDBC providers to use pureQuery in a Java SE environment.

Transaction support in WebSphere Application Server

Support for transactions is provided by the transaction service within WebSphere Application Server. The way that applications use transactions depends on the type of application component.

A transaction is unit of activity, within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, during the processing of an SQL COMMIT statement, the database manager atomically commits multiple SQL statements to a relational database. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a *logical unit of work* (LUW). If a transaction involves multiple resource managers, for example multiple database managers, an external transaction manager is required to coordinate the individual resource managers. A transaction that spans multiple resource managers is referred to as a *global transaction*. WebSphere Application Server is a transaction manager that can coordinate global transactions, can be a participant in a received global transaction, and can also provide an environment in which resource manager local transactions can run.

The way that applications use transactions depends on the type of application component, as follows:

- A session bean can use either container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (component-managed transactions where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) and application client components use component-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface, and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service (OTS) protocol or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through Java EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with databases, JMS queues, and JCA connectors through their *local transaction* support, when you do not require distributed transaction coordination.

IBM i Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances, you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:

- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to complete any updates, the one-phase commit resource does not have to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one or more two-phase commit resource providers and where *last participant support* is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the Java EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:
- Enterprise Information Systems that are accessed through a resource adapter, as described in the Java EE Connector Architecture.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. Resource adapter components of the Java EE connector architecture that include support for local transactions provide a LocalTransaction interface. The LocalTransaction interface enables applications to request that the resource adapter commits or rolls back RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, an external transaction manager is needed to coordinate the updates to all the resource managers in a global transaction.

Global transaction support is available to web and enterprise bean components and, with some limitations, to application client components. Enterprise bean components can be subdivided into two categories: beans that use container-managed transactions (CMT) and beans that use bean-managed transactions (BMT).

Only BMT enterprise beans, application client components, and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. To obtain the UserTransaction interface, use a Java Naming and Directory Interface (JNDI) lookup of java:comp/UserTransaction, or use the getUserTransaction method from the SessionContext object.

The UserTransaction interface is not available to CMT enterprise beans. If CMT enterprise beans attempt to obtain this interface, an exception is thrown, in accordance with the Enterprise JavaBeans (EJB) specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location that is appropriate for the EJB version.

WebSphere Application Server Version 4 and later releases bind the UserTransaction interface at the JNDI location that is specified in the EJB Version 1.1 specification. This location is java:comp/UserTransaction.

A web component or enterprise bean (CMT or BMT) can use additional interfaces that provide JTA support. These interfaces provide the transaction identity and a mechanism to receive notification of transaction completion. The interfaces include the TransactionSynchronizationRegistry interface, the ExtendedJTATransaction interface, and the UOWSynchronizationRegistry interface.

Local transaction containment

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

Unspecified transaction context is defined in the Enterprise JavaBeans specification, Version 2.0 and later. For example, see the specification for this technology.

An LTC is a bounded unit-of-work scope, within which zero or more resource manager local transactions (RMLT) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. By default, an LTC is local to a bean instance; it is not shared across beans, even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on an enterprise application component, such as an enterprise bean or servlet, whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example, at the end of the method dispatch. There is no programmatic interface to the LTC support; LTCs are managed exclusively by the container. The application deployer configures LTCs on individual application components, either web application or EJB, by using transaction attributes in the application deployment descriptor.

A local transaction containment (LTC) might be configured as part of an application component's deployment descriptor to be shareable across multiple application components, including web application components and enterprise beans that use container-managed transactions, so that those components can share connections without using a global transaction. Sharing a single resource manager between application components improves performance, increases scalability, and reduces lock contention for resources.

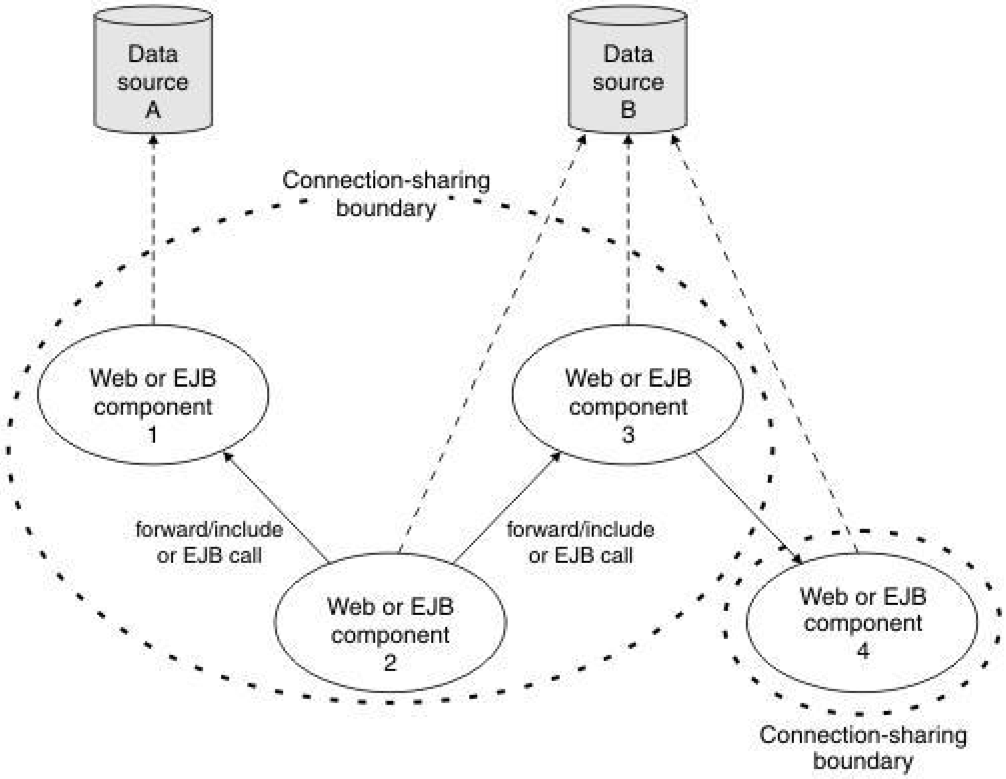
LTCs can be shared across multiple components, including web application components and enterprise beans that use container-managed transactions. This sharing is useful in situations such as frequent use of web component `include()` calls, where a thread can have several connections blocked by LTCs in different web modules. In this situation, the application might encounter code deadlocks under load, when threads start to wait for themselves to free connections. To overcome this issue without using a global transaction, specify that application components can share LTCs by setting the Shareable attribute in the deployment descriptor of each component. You must use a deployment descriptor; you cannot specify this attribute if annotation has been used.

When you set the Shareable attribute, the extended deployment descriptor XML file includes the following line of code:

```
<local-transaction boundary="BEAN_METHOD" resolver="CONTAINER_AT_BOUNDARY"
unresolved-action="COMMIT" shareable="true"/>
```

To obtain the full benefits of a shared LTC, also ensure that the resource reference for each component defaults to shareable connections.

In the following diagram, components 1, 2 and 3 are deployed with the Shareable attribute and component 4 is not. If components 2 and 3 both obtain connections to data source B, and their resource references for data source B default to shareable connections, they share the connection, but component 4 does not.



Applications that use shareable LTCs cannot explicitly commit or roll back resource manager connections that are used in a shareable LTC. Although, they can use connections that have an autoCommit capability. This ensures correct encapsulation of connection usage by each component and protects one component from having to make any assumptions about the behavior of other components that share the connection.

If an application starts any non-autocommit work in an LTC for which the Resolver attribute is set to `Application` and the Shareable attribute is set to `true`, an exception occurs at run time. For example, on a JDBC connection, non-autocommit work is work that the application performs after using the setAutoCommit(false) method to disable the autocommit option on the connection. Enterprise beans that use bean managed transactions (BMT) cannot be assembled with the Shareable attribute set on the LTC configuration.

IBM i A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for enterprise application components at J2EE 1.3 or later. The only exceptions to this are the following items:

- Application component dispatch occurs without container interposition, for example, for a stateless session bean create method or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 bean-managed transaction (BMT) enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that exists longer than the enterprise bean method in which it is started, as described in the topic about ActivitySessions and transaction contexts.

Local transaction containment

IBM WebSphere Application Server supports local transaction containment (LTC), which you can configure using local transaction extended deployment descriptors. LTC support provides certain advantages to application programmers. Use the scenarios provided, and the list of points to consider, to help you decide the best way to configure transaction support for local transactions.

The following sections describe the advantages that LTC support provides, and how to set the local transaction extended deployment descriptors in each situation.

You can develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not have to use global transactions, it is often more efficient to deploy the bean with the deployment descriptor for the container transaction type set to NotSupported instead of Required.

With the extended local transaction support of the application server, applications can perform the same business logic in an unspecific transaction context as they can in a global transaction. An enterprise bean, for example, runs in an unspecified transaction context if it is deployed with a container transaction type of NotSupported or Never.

The extended local transaction support provides a container-managed, implicit local transaction boundary, within which the container commits application updates and cleans up their connections. You can design applications with more independence from deployment concerns. This makes using a container transaction type of Supports much simpler, for example, when the business logic might be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage, regardless of whether the application runs in a transaction. The application can depend on the close action behaving in the same way in all situations, that is, the close action does not cause a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios, running business logic in a Transaction policy of NotSupported performs better than in a policy of Required. This benefit is applied through setting the deployment descriptor, in the Local Transactions section, of the Resolver attribute to ContainerAtBoundary. With this setting, application interactions with resource providers, such as databases, are managed within implicit resource manager local transactions (RMLT) that the container both starts and ends. The container commits RMLTs at the containment boundary that is specified by the Boundary attribute in the Local Transactions section; for example, at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

You can use local transactions in a managed environment that guarantees cleanup.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default setting of Application for the Resolver extended deployment descriptor in the Local Transactions section. In this situation, the container ensures connection cleanup at the boundary of the local transaction context.

Java platform for enterprise applications specifications that describe application use of local transactions do so in the manner provided by the default settings of Application for the Resolver extended deployment descriptor, and Rollback for the Unresolved action extended deployment descriptor, in the Local Transactions section. When the Unresolved action extended deployment descriptor in the Local Transactions section is set to Commit, the container commits any RMLTs that the application starts but that do not complete when the local transaction containment ends (for example, when the method ends). This usage applies to both servlets and enterprise beans.

IBM i **You can extend the duration of a local transaction beyond the duration of an EJB component method.**

The Enterprise JavaBeans (EJB) specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can use the Boundary extended deployment setting in the Local Transactions section to give the following advantages:

- Significantly extend the use cases of RMLTs.

- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

  You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which a client can control. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is applied through the Boundary extended deployment setting, in the Local transactions section, of ActivitySession. Such extended RMLTs can remain under the control of the application, or be managed by the container, depending on the setting of the Resolver deployment descriptor in the Local Transactions section.

You can coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can use ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans in that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

You can use shareable LTCs to reduce the number of connections you require.

Application components can share LTCs. If components obtain connections to the same resource manager, they can share that connection if they run under the same global transaction or shareable LTC. To configure two components to run under the same shareable LTC, set the Shareable attribute of the Local Transactions section in the deployment descriptor of each component. Make sure that the resource reference in the deployment descriptor for each component uses the default value of Shareable for the res-sharing-scope element, if this element is specified. A shareable LTC can reduce the numbers of RMLTs an application uses. For example, an application that makes frequent use of web module include calls can share resource manager connections between those web modules, exploiting either shareable LTCs, or a global transaction, reducing lock contention for resources.

Examples of local transaction support configurations

The following list gives scenarios that use local transactions, and points to consider when deciding the best way to configure the transaction support for an application.

- You want to start and end global transactions explicitly in the application (bean-managed transaction session beans and servlets only).

  For a session bean, set the Transaction type to Bean (to use bean-managed transactions) in the deployment descriptor of the component. You do not have to do this for servlets.
- IBM i You want to access only one XA or non-XA resource in a method.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to Supports.
- You want to access several XA resources atomically across one or more bean methods.

  In the deployment descriptor of the component, in the Container Transactions section, set the container transaction type to Required, RequiresNew, or Mandatory.
- IBM i You want to access several non-XA resources in a method without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to NotSupported.
- You want to access several non-XA resources in a method and want to manage them independently.

In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Unresolved action attribute to Rollback. In the Container Transactions section, set the container transaction type to NotSupported.

- IBM i You want to access one or more non-XA resources across multiple EJB method calls without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.
- IBM i You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.

Local and global transactions

Applications use resources, such as Java Database Connectivity (JDBC) data sources or connection factories, that are configured through the Resources view of the administrative console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider.

For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLT), but an XA data source can support two-phase commit coordination, as well as local transactions.

IBM i If an application uses two or more resource providers that support only RMLTs, atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application must use resources that support XA coordination and must access those resources in a global transaction.

If an application uses only one RMLT, atomic behavior can be guaranteed by the resource manager, which can be accessed in a local transaction containment (LTC) context.

An application can also access a single resource manager in a global transaction context, even if that resource manager does not support the XA coordination. An application can do this because the application server performs an “only resource optimization” and interacts with the resource manager in a RMLT. In a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but not both. An instance of an enterprise bean can change from running in one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions

Application clients can, within certain limits, support the use of transactions.

Application clients running in an enterprise application client container can explicitly demarcate transaction boundaries, as described in the topic about using component-managed transactions. Application clients cannot perform, directly in the client container, transactional work in the context of any global transaction that they start, because the client container is not a recoverable process.

Application clients can make requests to remote objects, such as enterprise beans, in the context of a client-initiated transaction. Any transactional work performed in a remote, recoverable, server process is coordinated as part of the client-initiated transaction. The transaction coordinator is created on the first server process to which the client-initiated transaction is propagated.

A client can begin a transaction, then, for example, access a JDBC data source directly in the client process. In such cases, any work performed through the JDBC provider is not coordinated as part of the global transaction. Instead, the work runs under a resource manager local transaction. The client container process is non-recoverable and contains no transaction coordinator with which a resource manager can be enlisted.

A client can begin a transaction, then call a remote application component such as an enterprise bean. In such cases, the client-initiated transaction context is implicitly propagated to the remote application server, where a transaction coordinator is created. Any resource managers accessed on the recoverable application server (or any other application server hosting application components invoked by the client) are enlisted in the global transaction.

Client application components must be aware that locally-accessed resource managers are not coordinated by client-initiated transactions. Client applications acknowledge this through a deployment option that enables access to the UserTransaction interface in the client container. By default, access to the UserTransaction interface in the client container is not enabled. To enable UserTransaction demarcation for an application client component, set the “Allow JTA Demarcation” extension property in the client deployment descriptor. For information about editing the client deployment descriptor, refer to the Rational Application Developer information.

Commit priority for transactional resources

You can specify the order in which transactional resources are processed during two-phase commit processing.

If you control the order in which transactional resources are processed during two-phase commit processing, there are two main benefits:

- One-phase commit optimization occurs more often.
- Potential problems caused by transaction isolation are resolved.

To control the order in which transactional resources are processed during two-phase commit processing, you specify the commit priority of a resource by setting the commit priority attribute on a resource reference. The larger the commit priority, the earlier the resource is processed. For example, if a resource has a commit priority of 10, it is processed before a resource with a commit priority of 1. The commit priority value is of type int and can be between -2147483648 and 2147483647.

If you do not specify a commit priority value, a default value of zero is assigned to the resource and is used when ordering resources at run time. If two or more resources are configured with the same priority, including the default priority, they are processed in an unspecified order with respect to each other.

You can specify the commit priority attribute on a resource reference by using Rational Application Developer tools. For detailed information, see the Rational Application Developer information center. The application component must have a deployment descriptor; you cannot specify this attribute if annotation has been used.

One-phase commit optimization

In a transaction with a two-phase commit, if every resource except the last one enlisted in the transaction votes read-only, indicating that those resources are not interested in the outcome of the transaction, a one-phase commit can occur. This means that the transaction service does not have to store resource and transaction information that it would need to roll back a two-phase commit, and therefore performance is improved.

You can control the order in which transactional resources are processed during two-phase commit, so you can process the resources that are most likely to vote read-only first. Therefore, you increase the chance that a one-phase commit might occur.

Typically, for a given transactional resource, you know the work that is performed at run time, so if you can control the order in which the resources in a transaction are processed, you can increase the likelihood of a one-phase commit optimization occurring.

Transaction isolation

When resources are involved in a global transaction, updates that are made as part of a transaction are not visible outside the transaction until the transaction commits, that is, those resources are isolated. This isolation can cause problems with other application components that act on the updates after they are committed. For example, further processing can fail, or can fail intermittently, because updates are order and time dependent. This problem does not occur with service integration bus messaging work in WebSphere Application Server, but can be a problem for other messaging providers, for example WebSphere MQ.

If you specify the order in which transactional resources are committed, problems caused by isolation are resolved for all transactional systems, not just messaging providers and service integration bus in particular.

The following example describes how problems might occur when you cannot specify the order in which transactional resources are committed. An application updates a row in a database table, then sends a JMS message that triggers additional processing of the row. Both of these actions are performed in the same global transaction, so they are isolated until their respective resources are committed. If the update to the row is committed before the message is sent, the processing that is triggered by the message can access the updated row and process it. If the action to send the message is committed first, this action might trigger the additional processing of the row before the database has committed the update to the row. In this situation, the updated row is still isolated and is not visible, so the additional processing of the row fails.

This problem can be more complicated because it is ordering and timing dependent. If the database is committed first, the problem does not occur. If the action to send the message is committed first, the problem might occur, but it depends whether the database work is committed before the message triggers the further processing of the row. Therefore, the problem can be intermittent, so it is harder to identify its cause.

Restrictions with earlier versions of WebSphere Application Server

If you specify the commit priority of a resource, that is, specify any value other than the default value 0, the commit priority is added to the partner log in a recoverable unit section. This section in the log file is recognized in WebSphere Application Server Version 7.0 or later, but not in earlier versions of the application server.

Therefore, if an application uses the commit priority attribute, you cannot install that application into a mixed-version cluster where one or more servers in the cluster are at versions of WebSphere Application Server that are earlier than Version 7.0.

Also, if an application that uses the commit priority attribute is installed in a cluster, you cannot subsequently add a server to that cluster if the server is at a version of WebSphere Application Server that is earlier than Version 7.0.

For general information about different versions of the product, see the topic “Overview of migration, coexistence, and interoperability”.

Transaction compensation and business activity support

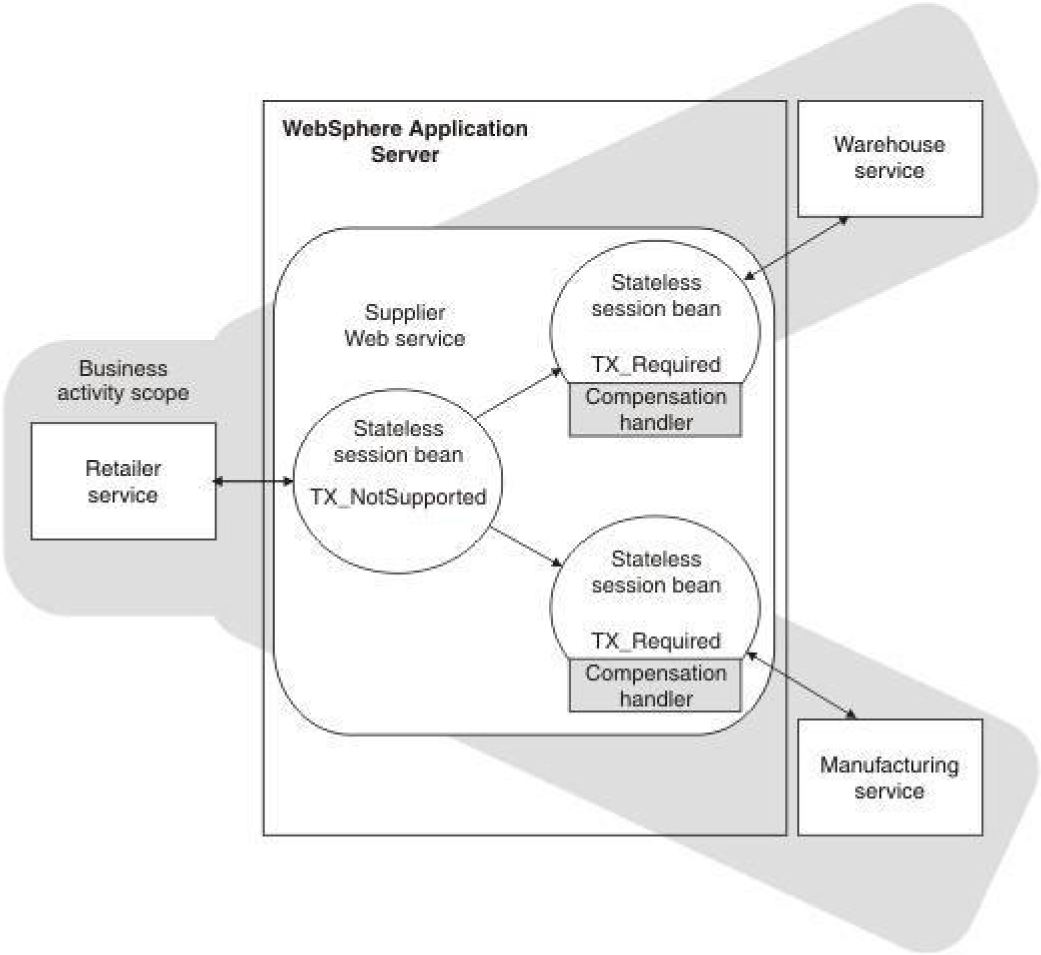
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an email can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Serverenvironments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that undertake work that can be compensated. These other session beans each have a *compensation handler*, a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an email.



Application design

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work by using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functions. Enterprise beans that exploit business activity scopes can offer web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in web service messages by using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

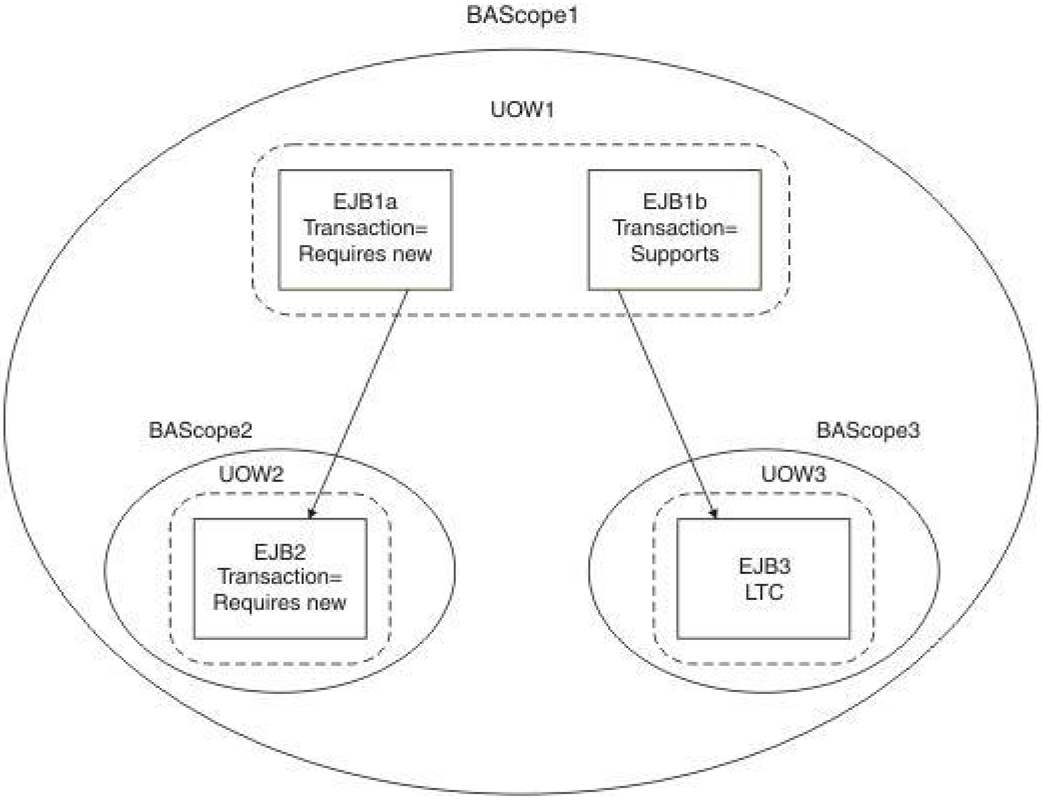
Business activity scopes

The scope of a business activity is that of a main WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing main UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any main UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.


BAScope1
UOW1
EJB1a Transaction= Requires new
EJB1b Transaction= Supports
BAScope2
UOW2
EJB2 Transaction= Requires new
BAScope3
UOW3
EJB3 LTC

Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by trying the called component again or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 19. Compensation behavior for a single business activity scope. The table lists the possible combinations of success and failure for the inner and outer business activity scopes, and the compensation behavior associated with each combination.

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see the topic about the business activity API.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight by using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration

If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope

Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the

received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method, then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For details, see the topic about creating an application that uses the WS-BA support.

JTA support

Java Transaction API (JTA) support provides application programming interfaces (APIs) in addition to the UserTransaction interface that is defined in the JTA 1.1 specification.

These interfaces include the TransactionSynchronizationRegistry interface, which is defined in the JTA 1.1 specification, and the following API extensions:
- SynchronizationCallback interface
- ExtendedJTATransaction interface
- UOWSynchronizationRegistry interface
- UOWManager interface

The APIs provide the following functions:
- Access to global and local transaction identifiers associated with the thread.

  The global identifier is based on the transaction identifier in the CosTransactions::PropagationContext: object and the local identifier identifies the transaction uniquely in the local Java virtual machine (JVM).
- A transaction synchronization callback that any enterprise application component can use to register an interest in transaction completion.

  Advanced applications can use this callback to flush updates before transaction completion and clear up state after transaction completion. Java EE (and related) specifications position this function typically as the domain of the enterprise application containers.

  Components such as persistence managers, resource adapters, enterprise beans, and web application components can register with a JTA transaction.

The following information is an overview of the interfaces that the JTA support provides. For more detailed information, see the generated API documentation.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run on a Java platform for enterprise applications server, there is no specific enterprise application component active when this object is called. So, the object has limited direct access to any enterprise application resources. Specifically, the object has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to an enterprise application component (for example, a stateless session bean) that it delegates to. The object would then have all the usual access to enterprise application resources. For example, you might use the object to acquire a Java Database Connectivity (JDBC) connection and flush updates to a database during the beforeCompletion method.

ExtendedJTATransaction interface

This interface is a WebSphere programming model extension to the Java EE JTA support. An object implementing this interface is bound, by enterprise application containers in WebSphere Application Server

that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an Enterprise JavaBeans (EJB) container, is not restricted to component-managed transactions.

An application uses a Java Naming and Directory Interface (JNDI) lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which the application uses as shown in the following example:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallback objects. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server, whether the transaction is started locally or imported
- At the end of the transaction for which the callback was registered

Note: In this release, the registerSynchronizationCallbackForCurrentTran method is deprecated. Use the registerInterposedSynchronization method of the TransactionSynchronizationRegistry interface instead.

TransactionSynchronizationRegistry interface

This interface is defined in the JTA 1.1 specification. System-level application components, such as persistence managers, resource adapters, enterprise beans, and web application components, can use this interface to register with a JTA transaction. Then, for example, the component can flush a cache when a transaction completes.

To obtain the TransactionSynchronizationRegistry interface, use a JNDI lookup of java:comp/TransactionSynchronizationRegistry.

Note: Use the registerInterposedSynchronization method to register a synchronization instance, rather than the registerSynchronizationCallbackForCurrentTran method of the ExtendedJTATransaction interface, which is deprecated in this release.

UOWSynchronizationRegistry interface

This interface provides the same functions as the TransactionSynchronizationRegistry interface, but applies to all types of units of work (UOWs) that WebSphere Application Server supports:

- JTA transactions
- local transaction containments (LTCs)
- ActivitySession contexts

System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface to register with a JTA transaction. The component can do the following:

- Register synchronization objects with special ordering semantics.
- Associate resource objects with the UOW.
- Get the context of the current UOW.
- Get the current UOW status.
- Mark the current UOW for rollback.

To obtain the UOWSynchronizationRegistry interface, use a JNDI lookup of java:comp/websphere/UOWSynchronizationRegistry. This interface is available only in a server environment.

The following example registers an interposed synchronization with the current UOW:

```
// Retrieve an instance of the UOWSynchronizationRegistry interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWSynchronizationRegistry uowSyncRegistry =
  (UOWSynchronizationRegistry)initialContext.lookup("java:comp/websphere/UOWSynchronizationRegistry");

// Instantiate a class that implements the javax.transaction.Synchronization interface
final Synchronization sync = new SynchronizationImpl();

// Register the Synchronization object with the current UOW.
uowSynchronizationRegistry.registerInterposedSynchronization(sync);
```

UOWManager interface

The UOWManager interface is equivalent to the JTA TransactionManager interface, which defines the methods that allow an application server to manage transaction boundaries. Applications can use the UOWManager interface to manipulate UOW contexts in the product. The UOWManager interface applies to all types of UOWs that WebSphere Application Server supports; that is, JTA transactions, local transaction containments (LTCs), and ActivitySession contexts. Application code can run in a particular type of UOW without needing to use an appropriately configured enterprise bean. Typically, the logic that is performed in the scope of the UOW is encapsulated in an anonymous inner class. System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface.

WebSphere Application Server does not provide a TransactionManager interface in the API or the system programming interface (SPI). The UOWManager interface provides equivalent functions, but WebSphere Application Server maintains control and integrity of the UOW contexts.

To obtain the UOWManager interface in a container-managed environment, use a JNDI lookup of java:comp/websphere/UOWManager. To obtain the UOWManager interface outside a container-managed environment, use the UOWManagerFactory class. This interface is available only in a server environment.

You can use the UOWManager interface to migrate a web application to use web components rather than enterprise beans, but maintain control over the UOWs. For example, a web application currently uses the UserTransaction interface to begin a global transaction, makes a call to a method on a session enterprise bean that is configured as not supported to undertake some non-transactional work, and then completes the global transaction. You can move the logic that is encapsulated in the session EJB method to the run method of a UOWAction implementation. Then, you replace the code in the web component that calls the session enterprise bean with a call to the runUnderUOW method of a UOWManager interface to request that this logic is run in a local transaction. In this way, you maintain the same level of control over the UOWs as you had with the original application.

The following example performs some transactional work in the scope of a new global transaction. The transactional work is performed in an anonymous inner-class that implements the run method of the UOWAction interface. Any checked exceptions that the run method creates do not affect the outcome of the transaction.

```
// Retrieve an instance of the UOWManager interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWManager uowManager = (UOWManager)initialContext.lookup("java:comp/websphere/UOWManager");

try
{
 // Invoke the runUnderUOW method, indicating that the logic should be run in a global
 // transaction, and that any existing global transaction should not be joined, that is,
 // the work must be performed in the scope of a new global transaction.
 uowManager.runUnderUOW(UOWSynchronizationRegistry.UOW_TYPE_GLOBAL_TRANSACTION, false, new UOWAction()
 {
   public void run() throws Exception
   {
```

```
   // Perform transactional work here.
   }
 });
}

catch (UOWActionException uowae)
{
 // Transactional work resulted in a checked exception being thrown.
}

catch (UOWException uowe)
{
 // The completion of the UOW failed unexpectedly. Use the getCause method of the
 // UOWException to retrieve the cause of the failure.
}
```

Chapter 9. Dynamic caching

This page provides a starting point for finding information about the dynamic cache service, which improves performance by caching the output of servlets, commands, web services, and JavaServer Pages (JSP) files.

Dynamic caching features include replication of cache entries, cache disk offload, Edge-Side Include caching, web services, and external caching. Use external caching to control caches outside of the application server.

Dynamic cache service eviction policies

Disk cache infrastructure enhancements

Several performance enhancements are available for the dynamic cache service.

The dynamic cache service supports persisting objects to disk (specified by a file system location) so that objects that are evicted from the memory cache are not regenerated by the application server. Objects are written to disk when they are evicted from memory using a Least Recently Used (LRU) eviction algorithm. The objects in the memory cache may also be flushed to disk on normal server shutdown. Java objects that need to be offloaded to the disk should be serializable.

The disk offload function includes the following functions:

- An internal disk cache format for faster deletions and support for new options to limit disk cache size
- The disk cache garbage collector, which evicts objects out of the cache when a configured high threshold is reached
- Four new performance modes to tune your disk cache performance:
  - High performance/memory usage mode - keeps all metadata in system memory and provides the highest performance
  - Balanced performance/memory usage mode - provides optimal balance of performance and memory usage by keeping some metadata in system memory
  - Custom performance/memory usage mode - allows explicit configuration of the memory usage and customization of performance requirements
  - Low performance/memory usage mode - stores most of the metadata on disk for users who are very constrained on system memory

Limiting the disk cache. The dynamic cache service provides mechanisms to limit the use of the disk cache by specifying the size of the disk cache in gigabytes, in addition to the maximum number of entries that are persisted to the disk. The disk cache is considered full when either of these limits is reached and forms the basis for eviction of objects from the disk. If the cache subsystem cannot offload any more data to disk, due to either an out-of-disk space condition, insufficient space on disk, or an exception when writing data to disk as a result of a possibly corrupt disk, the disk offload capability is disabled to prevent data integrity problems. The event is logged and the disk cache subsystem is deleted. This prevents serving corrupt data from the cache on a restart. If the option to persist cache data is turned on, some information such as dependency and template information is flushed to disk on a server shutdown. If a disk full situation occurs during this shutdown process, any partially-persisted and un-persisted dependency or template data is removed from the cache. A side effect of this, to preserve integrity, is to invalidate the cached objects that are associated with the dependency or template data.

Disk cache size in GB. The disk cache size in GB option pertains primarily to the object data (which includes the cached object, its identifier, and metadata such as expiration time), template information and dependency information that are written to disk. The cache subsystem allocates separate storage and volumes (each of which can grow to 1 GB) for object data, templates and dependencies, as needed.

When the total number of volumes on disk exceeds the specified cache size, any subsequent data that is written to disk is discarded until more space is made available by the disk cache garbage collector. To preserve data integrity, any information that is related to discarded objects is invalidated as well. The thresholds for garbage collection (described later in this document) and the disk cache full state are associated with the space available for object data. It is also possible that in certain, rare scenarios, as information is flushed to disk, critical system data needs to be written to disk, which may cause the total file system space required to exceed up to 5% of the specified maximum limit. It is recommended that there be at least 25% of actual file system space available for disk caching over and exceeds the specified disk cache size in GB. It is also required that each cache instance has a unique disk offload location and it is recommended that each offload location be on a dedicated disk partition. The cache file system employs a logical file manager to manage storage allocation for cached objects, therefore the file system size or the size of the files in the cache directory may not be an accurate gauge of the available space for the cache subsystem. At the same time, because of the adjusted limit, the cache subsystem may encounter a cache full state prior to the approaching the specified maximum limit as measured in allocated file system space. The PMI counters provide a better picture of how full the cache is.

Disabling the creation of ExtensionRegistry cache files. Whenever a servant restarts, a new ExtensionRegistry cache file is created in the dynacache directory. These files keep accumulating because they are never deleted. If you do not need the data that is collected in these cache files, you can add the `disable.dynacache.offload` property to the extension registry properties file, and set the value of this property to `true`. Setting this property to true disables the creation of ExtensionRegistry cache files for that server. The extension registry properties file is located at the server configuration level under each profile:

```
profile_home/config/cells/cell_name/nodes/node_name
      /servers/server_name/extensionregistry.properties
```

Eviction policies using the disk cache garbage collector

The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

The garbage collector keeps a certain amount of space on disk available, which is governed by the configuration attribute that limits the amount of disk space that is used for caching objects. To enable the eviction policy, enable the Limit disk cache size in GB and/or Limit disk cache size in entries options in the administrative console.

The garbage collector is triggered when the disk space reaches a specified high threshold (a percentage of the Limit disk cache size in entries or in GB) and evicts objects, based on the eviction policy, from the disk in the background until the disk cache size reaches a specified low threshold (a percentage of the Limit disk cache size in entries or in GB). Eviction triggers when one or both of the high thresholds is reached for Limit disk cache size in GB and Limit disk cache size in entries. The supported policies are:

- None: This is the default policy. Objects are evicted only when they expire, or if they are invalidated.
- Random: The expired objects are removed first. If the disk size still has not reached the low threshold limit, objects are picked from the disk cache in random order and removed until the disk size reaches a low threshold limit.
- Size: The expired objects are removed first. If the disk size still has not reached the low threshold limit, then largest-sized objects are removed until the disk size reaches a low threshold limit.

Limit disk cache size in GB and High Threshold determines when to trigger eviction and when the disk cache is considered near full. It is computed as a function of the user-specified limit. If the specified limit is *10* GB (3 GB is the minimum), the cache subsystem initially creates three files that can grow to 1 GB in size for cache data, dependency ID information, and template information. Each time more space is needed to contain cache data, dependency ID information, or template information, a new file is created. Each of these files grow in 1 GB increments until the total number of files that are created is equal to disk cache in size in GB (in this case ten). Although the initial size of the new file may be much smaller than 1 GB, the dynamic cache service always rounds up to the next GB.

Eviction triggers when the cache data size reaches the high threshold and continues until the cache data size reaches the low threshold. Calculation of cache data size is dynamic. The following formula describes how to calculate the actual cache data size limit:

```
cache data size limit = disk cache size (in GB) - number of dependency files per GB - number of template files
```

When the cache data size limit is defined, the trigger point is calculated as follows:

```
eviction trigger point = cache data size limit * high threshold
size of evicted entries = cache data size * (high threshold - low threshold)
```

Consider the following scenarios:

- **Scenario 1**
  - Disk cache size in GB = 10 GB
  - High threshold = 90%
  - Low Threshold = 80%

  Initially, there is one file for dependency ID and template ID.

  ```
  cache data size limit = 10-(1+1) = 8 GB
  eviction trigger point = 8 * 90% = 7.2 GB
  size of evicted entries = 8 * (90% - 80%) = 0.8 GB
  ```

  In the previous scenario, eviction starts when the data cache size reaches 7.2 GB and continues until the cache size is 6.4 GB (7.2 - 0.8).
- **Scenario 2**

  In scenario 1, if the dependency files grow to more than 1 GB, an additional dependency file generates. The eviction trigger point launches dynamically as follows:

  ```
  cache data size limit = 10 - (2+1) = 7GB
  eviction trigger point = 7 * 90% = 6.3GB
  size of evicted entries = 7 * (90% - 80%) = 0.7GB
  ```

  In the previous scenario, eviction starts when the data cache size reaches 6.3 GB, and continues until the cache size in 5.6 GB (6.3 - 0.7).

Disk cache eviction for limit disk cache size in entries. Consider the following scenario:

- Disk cache size in entries = 100000
- High threshold = 90%
- Low threshold = 80%

```
eviction trigger point = 100000 * 90% = 90000
number of entries evicted = 100000 * (90% - 80%) = 10000
```

In this scenario, eviction starts when the number of cache entries reaches 90000 and 10000 entries are evicted from the cache.

Example: Caching web services

This topic includes examples of building a set of cache policies and SOAP messages for a web services application.

The following is a example of building a set of cache policies for a simple web services application. The application in this example stores stock quotes and has operations to read, update the price of, and buy a given stock symbol.

Following are two SOAP message examples that the application can receive, with accompanying HTTP Request headers.

The first message sample contains a SOAP message for a GetQuote operation, requesting a quote for IBM. This is a read-only operation that gets its data from the back end, and is a good candidate for caching. In this example the SOAP message is cached and a timeout is placed on its entries to guarantee the quotes it returns are current.

Message example 1

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-lookup
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:getQuote xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

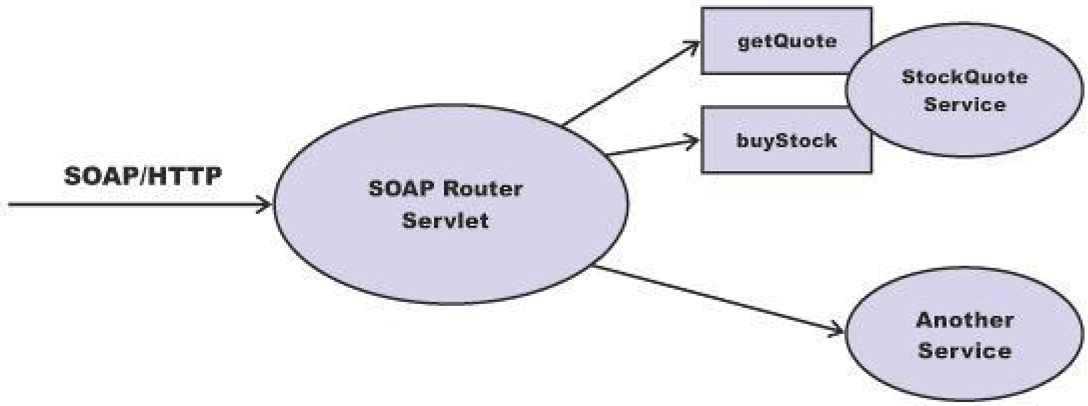
The SOAPAction HTTP header in the request is defined in the SOAP specification and is used by HTTP proxy servers to dispatch requests to particular HTTP servers. WebSphere Application Server dynamic cache can use this header in its cache policies to build IDs without having to parse the SOAP message.

Message example 2 illustrates a SOAP message for a BuyQuote operation. While message 1 is cacheable, this message is not, because it updates the back end database.

Message example 2

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-update
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:buyStock xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:buyStock>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following graphic illustrates how to invoke methods with the SOAP messages. In web services terms, especially Web Services Description Language (WSDL), a service is a collection of operations such as getQuote and buyStock. A body element namespace (urn:stockquote in the example) defines a service, and the name of the first body element indicates the operation.



The following is an example of WSDL for the getQuote operation:

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
targetNamespace="http://www.getquote.com/StockQuoteService-interface"
xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns=soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
<message name="SymbolRequest">
<part name="return" type="xsd:string"/>
</message>
<portType name="StockQuoteService">
<operation name="getQuote">
<input message="tns:SymbolRequest"/>
<output message="tns:QuoteResponse"/>
</operation>
</portType>
<binding name="StockQuoteServiceBinding"
type="tns:StockQuoteService">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getQuote">
<soap:operation soapAction="urn:stockquote-lookup"/>
<input>
<soap:body use="encoded" namespace="urn:stockquote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:stockquotes"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
</definition>
```

To build a set of cache policies for a web services application, configure WebSphere Application Server dynamic cache to recognize cacheable service operation of the operation.

WebSphere Application Server inspects the HTTP request to determine whether or not an incoming message can be cached based on the cache policies defined for an application. In this example, buyStock and stock-update are not cached, but stockquote-lookup is cached. In the `cachespec.xml` file for this web application, the cache policies need defining for these services so that the dynamic cache can handle both SOAPAction and service operation.

WebSphere Application Server uses the operation and the message body in web services cache IDs, each of which has a component associated with them. Therefore, each web services `<cache-id>` rule contains only two components. The first is for the operation. Because you can perform the stockquote-lookup operation by either using a SOAPAction header or a service operation in the body, you must define two different <cache-id> elements, one for each method. The second component is of type “body”, and defines how WebSphere Application Server should incorporate the message body into the cache ID. You can use a hash of the body, although it is legal to use the literal incoming message in the ID.

The incoming HTTP request is analyzed by WebSphere Application Server to determine which of the <cache-id> rules match. Then, the rules are applied to form cache or invalidation IDs.

The following is sample code of a cachespec.xml file defining SOAPAction and servicesOperation rules:

```
<cache>
<cache-entry>
 <class>webservice</class>
 <name>/soap/servlet/soaprouter</name>
 <sharing-policy>not-shared</sharing-policy>
 <cache-id>
  <component id="" type="SOAPAction">
   <value>urn:stockquote-lookup</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1<priority>
  </component>
 </cache-id>
 <cache-id>
  <component id="" type="serviceOperation">
   <value>urn:stockquote:getQuote</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1</priority>
  </component>
 </cache-id>
</cache-entry>
</cache>
```

Caching with Servlet 3.0

Dynamic cache provides servlet caching support for the Servlet 3.0 specification.

Be aware of the following API characteristics when using dynamic cache with Servlet 3.0:

- Dynamic cache wraps the ServletRequest and ServletResponse objects with its own cache application wrapper objects that extend ServletRequestWrapper and ServletResponseWrapper objects.
- Dynamic cache is always the first asyncListener added to the ServletRequest.
- Users of startAsync (ServletRequest req, ServletResponse res) and public AsyncContext startAsync() should flush the response before calling this method. Flushing the response ensures that any data that is written to the wrapped cache response is not lost.
- Do not read from or write to the request and response objects that are passed into public void addListener (asyncListener, req, res). Additional wrapping might have occurred since the given AsyncListener was registered, and might be used to release any resources that are associated with them.
- The do-not-consume property is not supported for Servlet 3.0 when using dynamic cache. The runtime forces the parent servlet to consume subfragments and the do-not-consume property is ignored.

Chapter 10. EJB applications

This page provides a starting point for finding information about enterprise beans.

Based on the Enterprise JavaBeans (EJB) specification, enterprise beans are Java components that typically implement the business logic of Java 2 Platform, Enterprise Edition (J2EE) applications as well as access data.

Enterprise beans

An enterprise bean is a Java component that can be combined with other resources to create Java applications. There are three types of enterprise beans, entity beans, session beans, and message-driven beans.

All beans reside in Enterprise JavaBeans (EJB) containers, which provide an interface between the beans and the application server on which they reside.

EJB 2.1 and earlier versions of the specification define entity beans as a means to store permanent data, so they require connections to a form of persistent storage. This storage might be a database, an existing legacy application, a file, or another type of persistent storage.

The EJB 3.0 specification deprecates EJB 1.1-style entity beans. The Java Persistence API (JPA) specification is intended to replace the deprecated enterprise beans. While the JPA replacement is called an entity class, it should not be confused with entity enterprise beans. A JPA entity is not an enterprise bean and is not required to run in an EJB container.

Session beans typically contain the high-level and mid-level business logic for an application. Each method on a session bean performs a particular high-level operation. For example, submitting an order or transferring money between accounts. Session beans often invoke methods on entity beans in the course of their business logic.

Session beans can be either stateful, stateless, or singleton. A stateful bean instance is intended for use by a single client during its lifetime, where the client performs a series of method calls that are related to each other in time for that client. One example is a shopping cart where the client adds items to the cart over the course of an online shopping session. In contrast, a stateless bean instance is typically used by many clients during its lifetime, so stateless beans are appropriate for business logic operations that can be completed in the span of a single method invocation. Stateful beans should be used only where absolutely necessary. Using stateless beans improves the ability to debug, maintain, and scale the application.

The EJB 3.1 specification introduces singleton session beans. The EJB container initializes only one instance of a singleton session bean, and that instance is shared by all clients. Because a single instance is shared by all clients, singleton session beans have special life cycle and concurrency semantics. Singleton session beans can have business local, business remote, and web service client views; they cannot have EJB 2.1 local or remote client views.

The EJB 3.x specifications support stateless and stateful session beans. They follow a simple pattern such as:

- Define the business interface.
- Define the class that implements it.
- Add metadata with annotations or with XML deployment descriptors.

The result of a simple EJB 3.x stateful session bean looks like the following:

```
package ejb3demo;

@Stateful
public class Cart3Bean implements ShoppingCart {
    private ArrayList contents = new ArrayList();

public void addToCart (Object o) {
  contents.add(o);
    }

public Collection getContents() {
    return contents;

    }
}
```

EJB components can use annotations such as @EJB and other injectable @Resource references if the module is an EJB 3.x module.

Web application clients and application clients can use deployment descriptor-defined EJB references. If the reference is for an EJB 3.x session bean without a home interface, the reference should be defined with a null <home> or <local-home> setting in the deployment descriptor.

Web application clients and application clients can also use @EJB injections for references to EJB session beans within the same enterprise archive (EAR) file, but the binding must either use the AutoLink support within the container or the annotation must use the name of the reference that is defined by the deployment descriptor and bound when the application is installed. For more information about AutoLink, see the topic, “EJB 3.x application bindings support”.

Message-driven beans enable asynchronous message servicing.

- The EJB container and a Java Message Service (JMS) provider work together to process messages. When a message arrives from another application component through JMS, the EJB container forwards it through an onMessage method call to a message-driven bean instance, which then processes the message. In other respects, message-driven beans are similar to stateless session beans.
- The EJB container and a Java Connector Architecture (JCA) resource adapter work together to process messages from an enterprise information system (EIS). When a message arrives from an EIS, the resource adapter receives the message and forwards it to a message-driven bean, which then processes the message. The message-driven bean is provided services such as transaction support by the EJB container in the same way that other enterprise beans are provided service.

Beans that require data access use data sources, which are administrative resources that define pools of connections to persistent storage mechanisms.

Java EE application resource declarations

You can configure your Java Enterprise Edition (Java EE) applications to declare dependencies on external resources and configuration parameters. These resources might be injected into the application code, or might be accessed by the application through the Java Naming and Directory Interface (JNDI).

Resource references allow an application to define and use logical names that you can bind to resources when the application is deployed.

The following resource types can be declared by Java EE applications: simple environment entries, Enterprise JavaBeans (EJB) references, web service references, resource manager connection factory references, resource environment references, message destination references, persistence unit references, and persistence context references.

Simple Environment Entries

You can define configuration parameters in your Java EE applications to customize business logic using simple environment entries. As described in the Java EE 6 application, simple environment entry values might be one of the following Java types: String, Character, Bye, Short, Integer, Long, Boolean, Double, Float, Class, and any subclass of Enum.

Note: The Java type, Class, and any subclass of Enum are new in Java EE 6.

The application provider must declare all of the simple environment entries accessed from the application code. The simple environment entries are declared using either annotations (javax.annotation.Resource) in the application code, or using env-entry elements in the XML deployment descriptor.

In the following example from an application, annotations declare environment entries:

```
// Retry interval in milliseconds
@Resource long retryInterval = 3000;
```

In the previous example, the field default value is 3000. You can use an env-entry-value, which you define in the XML deployment descriptor to change this value.

In the following example, an application declares a simple environment entry of type Class, and defines the Class to be injected using an env-entry-value element in the XML deployment descriptor.

```
@Resource(name=TraceFormatter) Class<?> traceFormatter;


<env-entry>
    <env-entry-name>TraceFormatter</env-entry-name>
    <env-entry-value>com.sample.trace.StdOutTraceFormatter</env-entry-value>
</env-entry>
```

In the previous example, the field value is set to the com.sample.trace.StdOutTraceFormatter Class object.

In the following example, an application which declares a simple environment entry called validationMode as a subclass of Enum in the com.sample.Order class, and configures the Enum value of CALLBACK to inject using elements in the XML deployment descriptor.

```
<env-entry>
    <env-entry-name>JPAValidation</env-entry-name>
    <env-entry-type>javax.persistence.ValidationMode</env-entry-type>
    <env-entry-value>CALLBACK</env-entry-value>
    <injection-target>
        <injection-target-class>com.sample.Order</injection-target-class>
        <injection-target-name>validationMode</injection-target-name>
    </injection-target>
</env-entry>
```

In the previous example, the validationMode field is set to the CALLBACK Enum value. Use the same approach when you use annotations and XML code to declare simple environment entries; for example:

```
@Resource (name=JPAValidation)
javax.persistence.ValidationMode validationMode;


<env-entry>
    <env-entry-name>JPAValidation</env-entry-name>
    <env-entry-value>CALLBACK</env-entry-value>
</env-entry>
```

Note: The simple environment entry support of the Java type, Class, and any subclass of Enum is new for Java EE 6. Previously, you might have developed your applications to declare these types as application resources using the resource-env-ref element in the XML deployment descriptor or using the javax.annotation.Resource annotation. For applications that were using these Java types with the javax.annotation.Resource annotation, the com.ibm.websphere.ejbcontainer.EE5Compatibility system property must be enabled. Without the EE5Compatibility system property, the binding-name element of the resource-env-ref element in the `ibm-ejb-jar-bnd.xml` file is ignored, since the data type is now treated as a simple environment entry and not a resource environment reference.

Note: The `<lookup-name>` deployment descriptor element and the `lookup` annotation attribute are new in `Java` EE 6. They specify the JNDI name of a referenced EJB or resource, relative to the java:comp/env naming context. If either is used in a simple environment entry, you cannot use an `<env-entry-value>` in the same `<env-entry>`.

Enterprise JavaBeans (EJB) References

As described in the Java EE 6 specification, you can develop your Java EE applications to declare references to enterprise bean homes or enterprise bean instances using logical names called EJB references.

When an application declares a reference to an EJB, the EJB that you reference will be resolved with one of the following techniques.

- Specify an EJB binding in the ibm-ejb-jar-bnd.xml file or ibm-web-bnd.xml file
- Specify an <ejb-link> element in ejb-jar.xml file or web.xml file
- Specify a beanName attribute on the javax.ejb.EJB annotation
- Specify a <lookup-name> element in ejb-jar.xml file or web.xml file
- Specify a lookup attribute on the javax.ejb.EJB annotation
- Locate an enterprise bean that implements the interface declared as the type of the EJB reference (referred to as AutoLink).

The EJB container attempts to resolve the EJB reference using the previous techniques in the order they are listed.

Note: If `<lookup-name>` or `lookup` is used in an EJB reference, you cannot use `<ejb-link>` or `beanName` in the same EJB reference.

Note: All of the following EJB reference examples assume the SampleCart bean has only a single interface. If the SampleCart bean had multiple interfaces, then add the following suffix to the end of the binding, <ejb-link> element, or beanName attribute : !com.sample.Cart.

In the following example, an application declares an EJB reference using an annotation, and provides a binding for resolution.

```
@EJB(name="Cart")
Cart shoppingCart;


<ejb-ref name="Cart" binding-name="java:app/SampleEJB/SampleCart"/>
```

In the following example, an application declares an EJB reference using an annotation, and provides an ejb-link element for resolution.

```
@EJB(name="Cart")
Cart shoppingCart;


<ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <ejb-link>SampleEJB/SampleCart</ejb-link>
</ejb-local-ref>
```

In the following example, an application declares an EJB reference using an annotation, and provides a lookup attribute for resolution, from the source bean com.sample.SourceBean.

```
@EJB(name="Cart" lookup="java:app/SampleEJB/SampleCart")
Cart shoppingCart;
```

The application could alternatively declare the EJB reference using the <lookup-name> element in the XML deployment descriptor, as in the following example.

```
<ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <lookup-name>java:app/SampleEJB/SampleCart</lookup-name>
    <injection-target>
    <injection-target-class>com.sample.SourceBean</injection-target-class>
    <injection-target-name>ShoppingCart</injection-target-name>
    </injection-target>
</ejb-local-ref>
```

In the following example, an application declares an EJB reference using an annotation, and provides a beanName attribute for resolution.

```
@EJB(name="Cart" beanName="SampleEJB/SampleCart")
Cart shoppingCart;
```

Resource Environment References

As described in the Java EE 6 specification, you can develop applications to declare references to administered objects that are associated with a resource, such as a Connecter CCI InteractionSpec instance, or other object types managed by the EJB container, including javax.transaction.UserTransaction, javax.ejb.EJBContext, javax.ejb.TimerServcie, org.omg.CORBA.ORB, javax.validation.Validator, javax.validation.ValidatorFactory, or javax.enterprise.inject.spi.BeanManager.

When an application declares a reference to an administered object, you must provide a binding to the administered object when the application is deployed. You can provide the binding using the administrative console when you deploy the application, or you can add the binding to the WebSphere binding XML file, `ibm-ejb-jar-bnd.xml` or `ibm-web-bnd.xml`.

In the following example, an application declares a resource environment reference, and provides a binding to the resource:

```
@Resource(name="jms/ResponseQueue")
Queue responseQueue;


<session name="StatelessSampleBean">
    <resource-env-ref name="jms/ResponseQueue" binding-name="Jetstream/jms/ResponseQueue"/>
</session>
```

The application could alternatively declare the resource environment reference using the lookup attribute, and not require a binding, as in the following example:

```
@Resource(name="jms/ResponseQueue", lookup="Jetstream/jms/ResponseQueue")
Queue responseQueue;


<resource-env-ref>
    <resource-env-ref-name>jms/ResponseBean</resource-env-ref-name>
    <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

When an application declares a reference to a container managed object type, a binding is not used. The container provides the correct instance of the referenced object. In the following example, an application declares a resource environment reference to a container-managed object:

```
@Resource
javax.validation.Validator validator;
```

Resource References to Resource References

A new lookup field on the @Resource annotation is added with Java EE 6. You can now declare a resource reference to a resource reference as shown in the following example:

```
@Resource(name="java:global/env/jdbc/ds1ref",
              lookup="java:global/env/jdbc/ds1",
              authenticationType=Resource.AuthenticationType.APPLICATION,
              shareable=false)
    DataSource ds1ref;
@Resource(name="java:global/env/jdbc/ds1refref",
              lookup="java:global/env/jdbc/ds1ref",
              authenticationType=Resource.AuthenticationType.APPLICATION,
              shareable=true)
    DataSource ds1refref;
```

The lookup uses the innermost nesting of references, which in this case is "java:global/env/jdbc/ds1ref".

Message-driven beans - automatic message retrieval

WebSphere Application Server supports the use of message-driven beans as asynchronous message consumers.

The following figure shows an incoming message being passed automatically to the onMessage() method of a message-driven bean that is deployed as a listener for the destination. The message-driven bean processes the message, in this case passing the message on to a business logic bean for business processing.

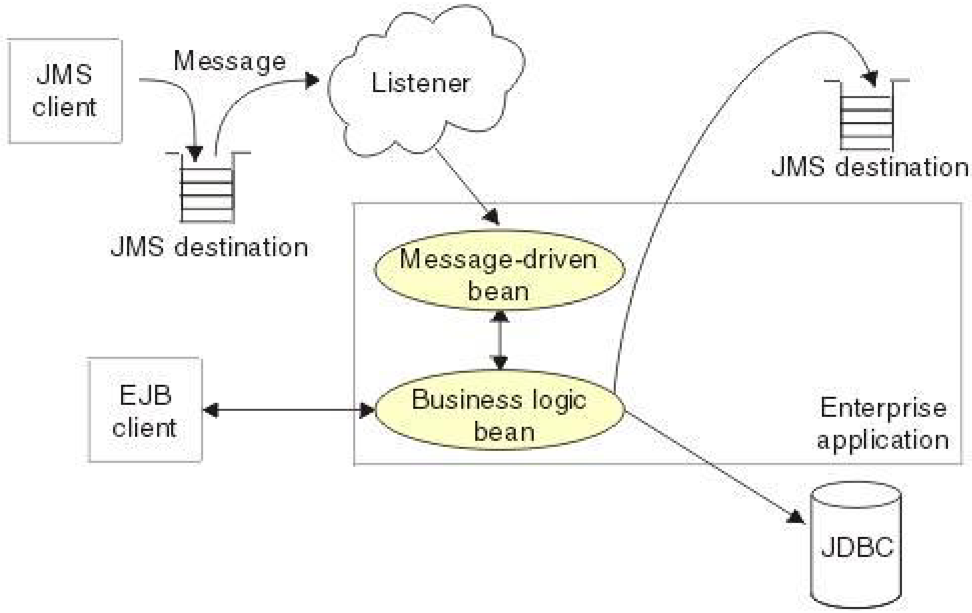


Figure 11. Messaging with message-driven beans

A client sends messages to the destination (or endpoint) for which the message-driven bean is deployed as the message listener. When a message arrives at the destination, the EJB container invokes the message-driven bean automatically without an application having to explicitly poll the destination. The message-driven bean implements some business logic to process incoming messages on the destination.

It can be helpful to separate the business logic of your application from the communication interfaces, such as the JMS request and response handling. To achieve this separation, you can design your message-driven bean to delegate the business processing of incoming messages to another enterprise bean. Separating message handling and business processing enables different users to access the same business logic in different ways, either through incoming messages or, for example, from a WebSphere J2EE client.

Message-driven beans can be configured as listeners on a Java EE Connector Architecture (JCA) 1.5 or 1.6 resource adapter or against a listener port. With a JCA 1.5 resource adapter, message-driven beans can handle generic message types, not just JMS messages. This makes message-driven beans suitable for handling generic requests inbound to WebSphere Application Server from enterprise information systems through the resource adapter. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or just *endpoints*.

All message-driven beans must implement the MessageDrivenBean interface. For JMS messaging, a message-driven bean must also implement the message listener interface, javax.jms.MessageListener.

Messages arriving at a destination being processed by a message-driven bean have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see Securing enterprise bean applications.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, for example the default messaging provider that is part of WebSphere Application Server or the WebSphere MQ messaging provider. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA 1.5-compliant resources, to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, for example the V5 default messaging provider, you must configure JMS message-driven beans against a listener port.

Message-driven beans, activation specifications, and listener ports

Guidelines, related to versions of WebSphere Application Server, to help you choose when to configure your message-driven beans to work with listener ports rather than activation specifications.

You can configure the following resources for message-driven beans:

- Activation specifications for message-driven beans that comply with Java EE Connector Architecture (JCA) Version 1.5.
- The message listener service, listener ports, and listeners for any message-driven beans that you want to deploy against listener ports.

Activation specifications are the standardized way to manage and configure the relationship between an MDB running in WebSphere Application Server and a destination in WebSphere MQ. They combine the configuration of connectivity, the Java Message Service (JMS) destination and the runtime characteristics of the MDB, within a single object.

Activation specifications supersede the use of listener ports, which became a stabilized feature in WebSphere Application Server Version 7.0 (for more information, see “Stabilized features” on page 1137). There are several advantages to using activation specifications over listener ports:

- Activation specifications are simple to configure, because they only require two objects: the activation specification and a message destination. Listener ports require three objects: a connection factory, a message destination, and the message listener port itself.
- Activation specifications are not limited to the server scope. They can be defined at any administrative scope in WebSphere Application Server. Message listener ports must be configured at the server scope. This means that each server in a node requires its own listener port. For example, if a node is made up

of three servers, three separate listener ports must be configured. Activation specifications can be configured at the node scope, so in the example only one activation specification would be needed.
- Activation specifications are part of the Java Platform, Enterprise Edition Connector Architecture 1.5 and 1.6 standards specification (JCA 1.5 and 1.6). Listener port support in WebSphere Application Server makes use of the application server facilities interfaces defined in the JMS specification, but is not part of any specification itself.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

If you want to use message-driven beans with a messaging provider that does not have a JCA Version 1.5 or 1.6 resource adapter, you cannot use activation specifications and therefore you must configure your beans against a listener port. There are also a few scenarios in which, although you could use activation specifications, you might still choose to use listener ports. For example, for compatability with existing message-driven bean applications. Here are some guidelines, related to versions of WebSphere Application Server, to help you choose when to use listener ports rather than activation specifications:
- WebSphere Application Server Version 4 does not support message-driven beans, so listener ports and activation specifications are not applicable. WebSphere Application Server Version 4 does support message beans, but these are not message-driven beans.
- WebSphere Application Server Version 5 supports EJB 2.0 (JMS only) message-driven beans that are deployed using listener ports. This deployment technology is sometimes called application server facility (ASF).
- WebSphere Application Server Version 6 continues to support message-driven beans that are deployed to use listener ports, and also supports JCA, which you can use to deploy message-driven beans that use activation specifications. This gives you the following options for deploying message-driven beans on WebSphere Application Server Version 6:
  - You must deploy default messaging (service integration bus) message-driven beans to use activation specifications.
  - You must deploy WebSphere MQ message-driven beans to use listener ports.
  - You can deploy third-party messaging message-driven beans to use either listener ports or activation specifications, depending on the facilities available from your third-party messaging provider.
- WebSphere Application Server Version 7.0 or later continues to support the same options for message-driven bean deployment that WebSphere Application Server Version 6 supports, and adds a new option for WebSphere MQ message-driven beans. This gives you the following options for deploying message-driven beans on Version 7.0 or later:
  - You must deploy default messaging (service integration bus) message-driven beans to use activation specifications.
  - You can deploy new and existing WebSphere MQ message-driven beans to use listener ports (as on WebSphere Application Server Version 6) or to use activation specifications.
  - You can deploy third-party messaging message-driven beans to use either listener ports or activation specifications, depending on the facilities available from your third-party messaging provider.

To assist in migrating listener ports to activation specifications, the WebSphere Application Server administrative console provides a **Convert listener port to activation specification** wizard on the Message listener port collection panel. This allows you to convert existing listener ports into activation specifications. However, this function only creates a new activation specification with the same configuration used by the listener port. It does not modify application deployments to use the newly created activation specification.

Message processing in ASF mode and non-ASF mode

Application Server Facilities (ASF) mode is the default method by which the message listener service in WebSphere Application Server processes messages. This topic explains how WebSphere Application Server processes messages in ASF mode and how it processes messages when ASF mode is turned off.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

Main features of ASF mode

By default, message-driven beans (MDBs) that are deployed on WebSphere Application Server for use with listener ports, use ASF mode to monitor JMS destinations and to process messages.

IBM i In ASF mode, a thread is allocated for work when a message is detected at the destination for it to process. The number of threads that can be active concurrently is dictated by the value specified for the **Maximum Sessions** property for the listener port.

IBM i In client connection (socket attach) mode, each active thread is an individual physical network connection. You should keep this in mind when you are deciding whether to use ASF or non-ASF mode in your configuration. If you are using WebSphere MQ Version 7.x as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

If WebSphere MQ is your messaging provider, there are several configurations you can use in ASF mode. With the following configurations each thread uses a separate physical network connection:

- A WebSphere MQ Version 6.0 queue manager.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `6`.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `0`.

With the following configuration, threads share a user-defined number of physical network connections:

- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `1` or higher. In this case each thread represents an individual connection to a queue manager. However, each thread does not have its own physical network connection, Instead, the threads share the number of network connections specified in the **SHARECNV** (sharing conversations) parameter.

IBM i

Main features of non-ASF mode

In non-ASF mode threads are active from the moment that the listener port is turned on. The number of active threads is dictated by the value specified for the **Maximum Sessions** property on the listener port. The number of threads specified in **Maximum Sessions** are active, regardless of the number of messages that are available to be processed.

In non-ASF mode, when a listener port browses for messages at the destination, it will take the message that is first in the queue at the destination for processing. This means that messages are processed close to the order in which they arrive at the destination.

In client connection (socket attach) mode, each active thread is an individual physical network connection. You should keep this in mind when you are deciding whether to use ASF or non-ASF mode in your configuration. If you are using WebSphere MQ Version 7.x as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

If WebSphere MQ is your messaging provider, there are several configurations you can use in non-ASF mode. With the following configurations each thread uses a separate physical network connection:

- A WebSphere MQ Version 6.0 queue manager.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to 6.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to 7 or unspecified, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to 0.

With the following configuration, threads share a user-defined number of physical network connections:

- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to 7 or unspecified, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to 1 or higher. In this case each thread represents an individual connection to a queue manager. However, each thread does not have its own physical network connection. Instead, the threads share the number of network connections specified in the **SHARECNV** (sharing conversations) parameter.

Note: Non-ASF mode cannot be selected on z/OS systems.

How messages are processed in ASF mode

In ASF mode, server sessions and threads are only allocated for work when a message that is suitable for the message-driven bean (MDB) is detected. The number of threads that an MDB can process concurrently is determined by the value of the **Maximum Sessions** property for the listener port.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

IBM i

The following diagram shows how messaging takes place between WebSphere Application Server and WebSphere MQ when the message listener service is operating in ASF mode.

Figure 12. Message processing in ASF mode on distributed and IBM i systems

As shown in the diagram, when the message listener service is operating in ASF mode, messages are processed in the following way:

1. When the listener port is started, it opens a connection to the WebSphere MQ queue manager and creates an internal queue agent.
2. The queue agent listens to the JMS destination for messages.
3. The queue agent detects a message.
4. The queue agent checks whether the message is suitable for the MDB that is using the listener port.
5. If the message is suitable for the MDB, the listener port allocates a thread from the message listener service thread pool, and allocates a server session from the application server's server session pool. If this is the first time the server session has been used since the listener port has been started, it opens a connection to the JMS provider. The allocated server session runs on the allocated thread.
6. The queue agent passes the ID of the message to the server session. It then starts listening for messages again.
7. The server session uses the message ID to retrieve the message from the destination.
8. The server session processes the message by calling the `onMessage()` method of the MDB.
9. When the message has been processed, the server session exits and returns to the application server session pool. The connection that the server session opened to the JMS provider remains open so that the server session does not need to re-establish the connection the next time it is used.
10. The thread exits and returns to the message listener service thread pool.

The number of threads that an MDB can process concurrently is determined by the value of the **`Maximum Sessions`** property for the listener port. If you set **`Maximum Sessions`** to the default value of `1`, this means that the MDB can only process one message at a time. Therefore, if the queue agent finds a second message before the first message has finished being processed, the queue agent blocks the second message until processing of the first message is complete and the server session has become available.

If you want to process more than one message concurrently, you can do this in ASF mode by setting **`Maximum Sessions`** to a value higher than `1`. For example, if you set **`Maximum Sessions`** to `2`, messages are processed in the following way:

1. The queue agent detects the first message and allocates a thread and a server session as in the first example. The message is processed using the `onMessage()` method of the MDB.
2. Whilst the first message is processing, the queue agent starts listening for messages again.
3. The queue agent detects the second message and allocates a second thread and a second server session. The message is processed using the `onMessage()` method of the MDB.
4. When the first message is processed, the first server session exits and returns to the server session pool. The first thread exits and returns to the thread pool.
5. When the second message is processed, the second server session exits and returns to the server session pool. The second thread exits and returns to the thread pool.

How messages are processed in non-ASF mode

IBM i

In non-ASF mode threads are active from the moment that the listener port is started. The number of active threads is dictated by the value specified for **`Maximum Sessions`**. The number of threads specified in **`Maximum Sessions`** are active, regardless of the number of messages that are available to be processed. Each active thread is an individual physical network connection.

If you are using WebSphere MQ Version 7.0 or later as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

Message processing in non-ASF mode

You activate non-ASF mode by specifying a non-zero value for the **NON.ASF.RECEIVE.TIMEOUT** message listener service custom property. **NON.ASF.RECEIVE.TIMEOUT** acts as a switch that turns off ASF mode, and also as a timeout value for the `receive()` method.

Note: The following message listener service custom properties do not work in non-ASF mode:

- **SERVER.SESSION.POOL.REAP**
- **SERVER.SESSION.POOL.UNUSED.TIMEOUT**
- **SERVER.SESSION.POOL.UNUSED.TIMEOUT.Ipaname**

The following diagram shows how message processing takes place between WebSphere Application Server and WebSphere MQ in non-ASF mode:

IBM i

Figure 13. Message processing in non-ASF mode

As shown in the diagram, when the message listener service is operating in non-ASF mode, messages are processed in the following way:

1. When the listener port is started, it gets one thread from the message listener service thread pool.
2. The listener port opens a connection to the WebSphere MQ queue manager on the thread and creates a JMS message consumer. The message consumer listens to the JMS destination which the listener port is configured to listen to.
3. The listener port creates a transaction to manage the message processing.
4. The thread calls the `receive()` method on the message consumer to listen for messages at the destination. If the `receive()` method does not detect a message in the time specified for **NON.ASF.RECEIVE.TIMEOUT**, the application server rolls back the active transaction and starts a new one. The thread then starts calling the `receive()` method again.
5. When the message consumer detects a message it checks whether the message is suitable for the MDB that is using the listener port.
6. If the message is suitable, the `receive()` method takes it off the destination and sends it to the thread.
7. The thread invokes the `onMessage()` method of the MDB on the message consumer, and the message is processed.
8. If the message finishes processing successfully, the transaction commits. If the message does not process successfully, the transaction rolls back.
9. A new transaction is started and the message consumer calls the `receive()` method to listen for new messages.

The number of threads that an MDB can process concurrently is determined by the value of the **Maximum Sessions** property for the listener port. If you set **Maximum Sessions** to the default value of `1`, this means that the MDB can only process one message at a time. If you want to process more than one message concurrently, you can do this in ASF mode by setting **Maximum Sessions** to a value higher than 1. For example, if you set **Maximum Sessions** to 2, messages are processed in the following way:

1. When the listener port is started, it gets two threads from the message listener service thread pool.
2. The listener port creates a message consumer and a transaction on each thread. The message consumers listen to the destination which the listener port is configured to listen to.
3. Both message consumers call the receive() method to listen for messages on the destination. The consumers compete to get messages from the destination.
4. When one of the consumers successfully retrieves the message, it processes it by calling the onMessage() method of the MDB. The other message consumer keeps on calling the receive() method to listen for messages on the destination.

How to avoid unwanted transaction timeouts

If your messaging system is running in non-ASF mode, to avoid unwanted transaction timeouts, you must allow a sufficient amount of time for processing to be completed before the total transaction lifetime timeout is reached. Therefore, you must make sure that the value that you specify for the **NON.ASF.RECEIVE.TIMEOUT** message listener service custom property is smaller than the value that you specify for the **Total transaction lifetime timeout** transaction service property, and also that the difference between the values of the two properties is greater than the amount of time that the onMessage() method of the message-driven bean (MDB) takes to process the message.

As the following example shows, if these properties are not correctly configured, transactions can time out before they are completed. This is because the thread begins calling the receive() method as soon as the transaction is created. In the following example, **NON.ASF.RECEIVE.TIMEOUT** is set to 110000 milliseconds (110 seconds), **Total transaction lifetime timeout** is set to 120 seconds and the onMessage () method of the MDB takes 15 seconds to process a message. The example supposes that a message does not appear at the destination until the receive() method has almost timed out:

1. The listener port starts. It allocates a thread from the thread pool and creates a transaction and a message consumer on the thread.
2. The thread calls the receive() method to listen for messages.
3. After 110 seconds a message appears at the destination.
4. The thread removes the message from the destination and calls the onMessage() method of the MDB to begin processing the message.
5. 10 seconds later, the transaction timeout is reached. The application server marks the transaction for rollback.
6. 5 seconds later, the onMessage() method finishes processing the message and tries to commit the transaction.
7. The total amount of time that has elapsed since the transaction was started is 125 seconds (110 seconds waiting for a message, plus 15 seconds to process the message). As this is longer than the transaction timeout, the application server prevents the transaction from being committed, and it is rolled back.

For further information about how to configure the **NON.ASF.RECEIVE.TIMEOUT** and **Total transaction lifetime timeout** properties to avoid unwanted transaction time outs, see the related tasks.

Message-driven beans - JCA components

There are several administrative components that you configure for message-driven beans as listeners on a Java EE Connector Architecture (JCA) 1.5 resource adapter.

Components for a JCA resource adapter

When a resource adapter is installed, it provides definitions and classes for administered objects such as activation specifications. The administrator creates and configures activation specifications with Java Naming and Directory Interface (JNDI) names that are then available for applications to use.

The JCA resource adapter uses an activation specification to configure a particular endpoint. Each application that configures one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application uses the activation specification to provide configuration properties for the processing of inbound messages.

JMS components used with a JCA messaging provider

Message-driven beans that implement the javax.jms.MessageListener interface can be used with JMS messaging.

An application that uses JMS messaging needs access at runtime to configured objects such as connection factories and destinations:

- When the JMS provider is the default JMS provider or the WebSphere MQ messaging provider, the administrator configures these objects for the JMS provider. For example, to configure a JMS activation specification for the WebSphere MQ messaging provider, in the WebSphere Application Server administrative console navigate to **Resources** > **JMS->Activation specifications**.
- Otherwise the administrator configures these objects for the JMS resource adapter, which connects the application to a JMS provider, by navigating to **Resources** > **Resource Adapters**.

If the application contains one or more message-driven beans, the administrator must configure either a JMS activation specification or a message listener port. For JCA-compliant messaging providers, the administrator usually configures an activation specification. But for the WebSphere MQ messaging provider there is a choice; the administrator can configure an activation specification or, for compatibility with previous versions of WebSphere Application Server, the administrator can configure a message listener port.

The JMS activation specification provides the deployer with information about the configuration properties of a message-driven bean related to the processing of the inbound messages. For example, a JMS activation specification specifies the name of the service integration bus to connect to, information about the message acknowledgement modes, message selectors, destination types, and whether durable subscriptions are shared across connections with members of a server cluster.

The activation specification identifies a JMS destination by specifying its JNDI name. The message-driven bean acts as a listener on a specific JMS destination.

The JMS destination refers to a service integration bus destination (or WebSphere MQ destination) that the administrator must also configure. For more information about JMS resources and service integration, see “Default messaging” on page 350.

J2C activation specification configuration and use

Configure J2C activation specifications, and use them in the deployment of message-driven beans for JCA 1.5 resources.

J2C activation specifications are part of the configuration of inbound messaging support that can be part of a JCA 1.5 resource adapter. Each JCA 1.5 resource adapter that supports inbound messaging defines one or more types of message listener in its deployment descriptor (`messagelistener` in the ra.xml). The message listener is the interface that the resource adapter uses to communicate inbound messages to the message endpoint. A message-driven bean (MDB) is a message endpoint and implements one of the message listener interfaces provided by the resource adapter. By allowing multiple types of message listener, a resource adapter can support a variety of different protocols. For example, the interface javax.jms.MessageListener, is a type of message listener that supports JMS messaging. For each type of message listener that a resource adapter implements, the resource adapter defines an associated activation specification (`activationspec` in the ra.xml). The activation specification is used to set configuration properties for a particular use of the inbound support for the receiving endpoint.

When an application containing a message-driven bean is deployed, the deployer must select a resource adapter that supports the same type of message listener that the message-driven bean implements. As part of the message-driven bean deployment, the deployer needs to specify the properties to set on the J2C activation specification. Later, during application startup, a J2C activation specification instance is created, and these properties are set and used to activate the endpoint (that is, to configure the resource adapter inbound support for the specific message-driven bean).

Applications with message-driven beans can also specify all, some, or none of the configuration properties needed by the ActivationSpec class, to override those defined by the resource adapter-scoped definition. These properties, specified as activation-config properties in the deployment descriptor for the application, are configured when the application is assembled. To change any of these properties requires redeploying the application. These properties are unique to this applications use and are not shared with other message-driven beans. Any properties defined in the application deployment descriptor take precedence over those defined by the resource adapter-scoped definition. This allows application developers to choose the best defaults for their applications.

Message-driven beans - transaction support

Message-driven beans can handle messages on destinations (or endpoints) within the scope of a transaction.

Transaction handling when using the Message Listener Service with WebSphere MQ JMS

There are three possible cases, based on the message-driven bean deployment descriptor setting you choose: container-managed transaction (required), container-managed transaction (not supported), and bean-managed transaction.

In the message-driven bean deployment descriptor settings, you can choose whether the message-driven bean manages its own transactions (bean-managed transaction), or whether a container manages transactions on behalf of the message-driven bean (container-managed transaction). If you choose container-managed transactions, in the deployment descriptor notebook, you can select a container transaction type for each method of the bean to determine whether container transactions are required or not supported. The default container transaction type is required.

Container-managed transaction (required)

In this case, the application server starts a global transaction before it reads any incoming message from the destination, and before the `onMessage()` method of the message-driven bean is invoked by the application server. This means that other EJBs that are invoked in turn by the message, and interactions with resources such as databases can all be scoped inside this single global transaction, within which the incoming message was obtained.

If this application flow completes successfully, the global transaction is committed. If the flow does not complete successfully, (if the transaction is marked for rollback or if a runtime exception occurs), the transaction is rolled back, and the incoming message is rolled back onto the message-driven bean destination.

Container-managed transaction (not supported)

In this case there is no global transaction, but the JMS provider can still deliver a message from a message-driven bean destination to the application server in a unit of work. You can consider this as a local transaction, because it does not involve other resources in its transactional scope.

The application server acknowledges message delivery on successful completion of the `onMessage()` dispatch of the message-driven bean (using the acknowledgement mode specified by the assembler of the message-driven bean).

However, the application server does not perform an acknowledge, if an unchecked runtime exception is thrown from the `onMessage()` method. So, does the message roll back onto the message-driven bean destination (or is it acknowledged and deleted)?

The answer depends on whether a syncpoint is used by your JMS provider and can vary depending on the operating platform (in particular the z/OS® operating platform can impart different behavior here).

If your JMS provider establishes a syncpoint around the message-driven bean message consumption in this container-managed transaction (not supported) case, the message is rolled back onto the destination after an unchecked exception.

If a syncpoint is not used, then the message is deleted from the destination after an unchecked exception.

For related information, see the technote “MDB behavior is different on z/OS than on distributed when getting nonpersistent messages within syncpoint” at http://www.ibm.com/support/docview.wss?uid=swg21231549.

Bean-managed transaction

In this case, the action is similar to the container-managed transaction (not supported) case. Even though there might be a user transaction in this case, any user transaction started within the onMessage dispatch of the message-driven bean does not include consumption of the message from the message-driven bean destination within the transaction scope. To do this, use the container-managed transaction (required) scenario.

Message redelivery

In each of the previous three cases, a message that is rolled back onto the message-driven bean destination is eventually re-dispatched. If the original rollback was due to a temporary system problem, you would expect the re-dispatch of the message-driven bean with this message to succeed on re-dispatch. If, however, the rollback was due to a specific message-related problem, the message would repeatedly be rolled back and re-dispatched. This is known as a poison message scenario.

If your messaging system uses listener ports, the application server handles this scenario, by tracking the frequency with which a specific message is dispatched, and by stopping the associated listener port after a specified number of attempted re-deliveries of that message have occurred.

If your messaging system uses listener ports, you can avoid a poison message scenario by configuring the following property:

`Maximum Retries`

The **`Maximum Retries`** parameter specifies the number of times that the listener tries to deliver a specific message to a message-driven bean instance before the listener is stopped.

If this parameter is set to `0` the listener port will stop after a single failure of a message to be delivered successfully.

For more information about this property, see Listener port settings.

If your messaging system uses activation specifications, the poison message scenario is handled in a slightly different way. Whereas listener ports track the number of times a specific message has failed and been re-delivered, activation specifications count the number of sequential message delivery failures.

If your messaging system uses the default messaging provider (service integration), you must configure the following properties on your activation specification to avoid a poison message scenario:

Automatically stop endpoints on repeated message failure
: You must make sure that this option is selected.

  This property suspends message delivery to the endpoint when the **Sequential failed message threshold** is reached.

Sequential failed message threshold
: This parameter determines how many message deliveries can fail before message delivery is suspended.

  To enable this parameter you must have the **Automatically stop endpoints on repeated message failure** option selected.

Delay between failing message retries
: This parameter specifies how much time must elapse before a message which has failed to be delivered successfully, is re-delivered.

  If you specify 0 for this parameter there will be no delay before a message is re-delivered.

  To enable this parameter you must have the **Automatically stop endpoints on repeated message failure** option selected.

For more information on these properties, see JMS activation specification [Settings].

If your messaging system uses the WebSphere MQ messaging provider, you must configure the following properties on your activation specification to avoid a poison message scenario:

Stop endpoint if message delivery fails
: You must make sure that this option is selected.

  This property suspends message delivery to the endpoint when the **Number of sequential delivery failures before suspending endpoint** is reached.

Number of sequential delivery failures before suspending endpoint
: This parameter determines how many message deliveries can fail before message delivery is suspended.

  To enable this parameter you must have the **Stop endpoint if message delivery fails** option selected.

For more information on these properties, see WebSphere MQ messaging provider activation specification advanced properties

As an alternative to relying on your application server to stop the listener port or activation specification if a poison message scenario occurs, you can configure WebSphere MQ to resolve the problem. In WebSphere MQ specify a backout queue (**BOQUEUE**), and a backout threshold value (**BOTHRESH**). If you do this, WebSphere MQ handles the poison message. For more information about handling poison messages, see the WebSphere MQ *Using Java* section of the WebSphere MQ library.

Message-driven beans - listener port components

The WebSphere Application Server support for message-driven beans deployed against listener ports is based on JMS message listeners and the message listener service, and builds on the application server facility (ASF) support in the JMS provider.

Note: From WebSphere Application Server Version 7, listener ports are stabilized. For more information, read the article on stabilized features. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the WebSphere MQ messaging provider, refer to related tasks.

The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

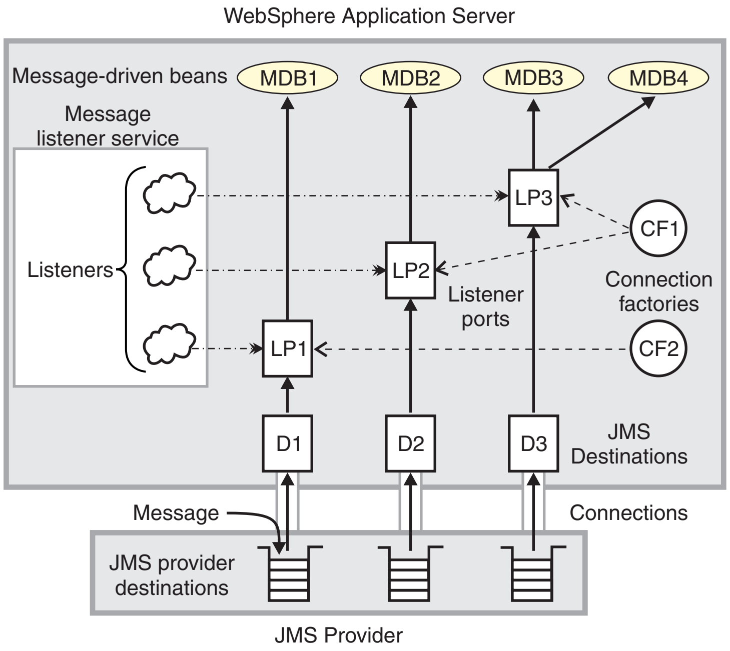


Figure 14. The main components for message-driven beans

The message listener service is an extension to the JMS functions of the JMS provider and provides a listener manager, which controls and monitors one or more JMS listeners. Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A *connection factory* is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A listener port defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports are used to simplify the administration of the associations between these resources.

When you deploy a message-driven bean, you associate the bean with a listener port. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the message listener service based on the configuration data. The message listener service creates a dynamic session thread pool for use by listeners, creates and starts listeners, and during server termination controls the cleanup of message listener service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:
- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.

- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

Access intent policies for EJB 2.x entity beans

An access intent policy is a named set of properties or access intents that govern data access for Enterprise JavaBeans (EJB) persistence. You can assign policies to an entity bean and to individual methods on an entity bean's home, remote, or local interfaces during assembly. You can set access intents only within EJB Version 2.x-compliant and later modules for entity beans with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic and update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run time environment.

transition: Access intent policies are specifically designed to supplement the use of isolation level and access intent method-level modifiers found in the extended deployment descriptor for EJB version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB version 2.x and later enterprise beans.

Access intent policies configured on an entity basis define the default access intent for that entity. The default access intent controls the entity unless you specify a different access intent policy based on either method-level configuration or application profiling.

Note: Method level access intents were deprecated in Version 6.x.

You can use application profiling or method level access intent policies to control access intent more precisely. Method-level access intent policies are named and defined at the module level. A module can have one or many policies. Policies are assigned, and apply, to individual methods of the declared interfaces of entity beans and their associated home interfaces. A method-based policy is acted upon by the combination of the EJB container and persistence manager when the method causes the entity to load.

For entity beans that are backed by tables with nullable columns, use an optimistic policy with caution. The top down default mapping excludes nullable fields. You can override this when doing a meet-in-the-middle mapping. The fields used in overqualified updates are specified in the ejb-rdb mapping. If nullable columns are selected as overqualified columns, partial update should also be selected.

An entity that is configured with a read-only policy that causes a bean to be activated can cause problems if updates are attempted within the same transaction. Those changes are not committed, and the process displays an exception because data integrity might be compromised.

Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time that a given resource is unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately after. Update locks are obtained immediately before an update operation and held until the end of the transaction.

To enable optimistic concurrency, this product uses an *overqualified update scheme* to test if the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back: All work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection.

best-practices: Whether to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. A high-penalty transaction is one for which recovery is risky or resource-intensive. For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often.

Read ahead scheme hints

read ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of container-managed persistence (CMP) beans for the transaction within one query. read ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that an application most likely needs next. A *read ahead hint* is a representation of the related beans to read. The hint is associated with the *findByPrimaryKey* method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.

A read ahead hint takes the form of a character string. You do not have to provide the string; the wizard generates it based on the container-managed relationships (CMRs) that are defined for the bean. The example is provided later in this section is for supplemental information only. Suppose a CMP bean type A has a finder method that returns instances of bean A. A read ahead hint for this method is specified using the following notation: `RelB.RelC; RelD`

Interpret the preceding notation as follows:
- Bean type A has a CMR with bean types B and D.
- Bean type B has a CMR with bean type C.

For each bean of type A that is retrieved from the database, its directly-related B and D beans and its indirectly-related C beans are also retrieved. The order of the retrieved bean data columns in each row of the result set is the same as the order in the read ahead hint: an A bean, a B bean (or null), a C bean (or null), a D bean (or null). For hints in which the same relationship is mentioned more than once, for example, `RelB.RelC;RelB.RelE`, the data columns for a bean occur only once in the result set, at the position the bean first occupies in the hint.

The tokens shown in the notation, like `RelB`, must be CMR field names for the relationships, as defined in the deployment descriptor for the bean. In indirect relationships such as `RelB.RelC`, `RelC` is a CMR field name that is defined in the deployment descriptor for bean type B.

A single read ahead hint cannot refer to the same bean type in more than one relationship. For example, if a Department bean has an *employees* relationship with the Employee bean and also has a *manager* relationship with the Employee bean, the read ahead hint cannot specify both *employees* and *manager*.

For more information about how to set read ahead hints, see the documentation for the Rational Application Developer product.

Run-time behaviors of read ahead hints

When developing your read ahead hints, consider the following tips and limitations:

- read ahead hints on long or complex paths can result in a query that is too complex to be useful. read ahead hints on root or leaf inheritance mappings need particular care. Add up the number of tables that potentially comprise a read ahead preload to gauge the complexity of the join operations that are required. Consider if the resulting statement constitutes a reasonable query on your target database.
- read ahead hints do not work in the following cases:
  - Preload paths across M:N relationships
  - Preload paths across recursive enterprise bean relationships or recursive fk relationships
  - When a read-head hint applies to a SELECT FOR UPDATE statement that requires a table join in a database that does not support the combination of those two operations.

    Generally, the persistence manager issues a SELECT FOR UPDATE statement for a bean only if the bean has an access intent that enforces strict locking policies. Strict locking policies require SELECT FOR UPDATE statements for database select queries. If the database table design requires a join operation to fulfill the statement, many databases issue exceptions because these databases do not support table joins with SELECT FOR UPDATE statements. In those cases, WebSphere Application Server does not implement a read ahead hint. If the database does provide that support, Application Server implements the read ahead hints that you configure.

Database deadlocks caused by lock upgrades

To avoid database deadlocks caused by lock upgrades, you can change the access intent policy for entity beans from the default of wsPessimisticUpdate-WeakestLockAtLoad to wsPessimisticUpdate, or you can use an optimistic locking approach.

When concurrently accessing data, ensure that the application is prepared for database locking that must occur to secure the integrity of the data.

If an entity bean performs a findByPrimaryKey method, which by default obtains a *Read* lock in the database, and the entity bean is updated within the same transaction, a lock upgrade to *Exclusive*.

If this scenario occurs concurrently on multiple threads, a deadlock can happen. This is because multiple read locks can be obtained at the same time but one exclusive lock can only be obtained when the other locks are dropped. Since all transactions are attempting the lock upgrade in this scenario, the one exclusive lock cannot be obtained.

To avoid this problem, you can change the access intent policy for the entity bean from the default of wsPessimisticUpdate-WeakestLockAtLoad method to wsPessimisticUpdate method. This change enables the application to inform the product and the database that the transaction has updated the enterprise bean. The *Update* lock is immediately obtained on the findByPrimaryKey method. This avoids the lock upgrade when the update is performed at a later time.

The preferred technique to define access intent policies is to change the access intent for the entire entity bean. You can change the access intent for the findByPrimaryKey method, but this was deprecated in Version 6. You might want to change the access intent for an individual method if, for example, the entity bean is involved in some transactions that are read only.

An alternative technique is to use an optimistic approach, where the findByPrimaryKey method does not hold a read lock, so there is no lock upgrade. However, this requires that the application is coded for this in order to handle rollbacks. Optimistic locking is intended for applications that do not expect database contention on a regular basis.

To change the access intent policy for an entity bean, you can use the assembly tool to set the bean level, as described in Applying access intent policies to beans.

Access intent assembly settings

Access intent policies contain data-access settings for use by the persistence manager. Default access intent policies are configured on the entity bean.

These settings are applicable only for EJB 2.x and EJB 3.x-compliant entity beans that are packaged in EJB 2.x and EJB 3.x-compliant modules. Connection sharing between beans with bean-managed persistence and those with container-managed persistence is possible if they all use the same access intent policy.

Name

Specifies a name for a mapping between an access intent policy and one or more methods.

Description

Contains text that describes the mapping.

Methods - name

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all of the methods of an enterprise bean's remote and home interfaces.

Methods - enterprise bean

Specifies which enterprise bean contains the methods indicated in the Name setting.

Methods - type

Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use `Unspecified` if an access intent policy applies to all methods of the bean.

Information	Value
Data type	String
Range	Valid values are `Home`, `Remote`,`Local`, `LocalHome` or `Unspecified`

Methods - parameters

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Applied access intent

Specifies how the container must manage data access for persistence. Configurable both as a default access intent for an entity and as part of a method-level access intent policy.

Information	Value
Data type	String
Default	`wsPessimisticUpdate-WeakestLockAtLoad`. With Oracle, this is the same as wsPessimisticUpdate.

Information	Value
Range	Valid settings are `wsPessimisticUpdate`, `wsPessimisticUpdate-NoCollision`, `wsPessimisticUpdate-Exclusive`, `wsPessimisticUpdate-WeakestLockAtLoad`, `wsPessimisticRead`, `wsOptimisticUpdate`, or `wsOptimisticRead`. Only `wsPessimisticRead` and `wsOptimisticRead` are valid when class-level caching is enabled in the EJB container.

This product supports lazy collections. For each segment of a collection, iterating through the collection (*next()*) does not trigger a remote method call to retrieve the next remote reference. Two policies (`wsPessimisticUpdate` and `wsPessimisticUpdate-Exclusive`) are extremely lazy; the collection increment size is set to 1 to avoid overlocking the application. The other policies have a collection increment size of 25.

If an entity is not configured with an access intent policy, the runtime environment typically uses `wsPessimisticUpdate-WeakestLockAtLoad` by default. If, however, the **Lifetime in cache** property is set on the bean, the default value of **Applied access intent** is `wsOptimisticRead`; updates are not permitted.

Additional information about valid settings follows:

Table 20. Access intents profiles. Here is additional information about valid settings:

Profile name	Concurrency control	Access type	Transaction isolation
wsPessimisticRead (Note 1)	pessimistic	read	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate (Note 2)	pessimistic	update	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate-Exclusive (Note 3)	pessimistic	update	serializable
wsPessimisticUpdate-NoCollision (Note 4)	pessimistic	update	read committed
wsPessimisticUpdate-WeakestLockAtLoad (Note 5)	pessimistic	update	Repeatable read
wsOptimisticRead	optimistic	read	read committed
wsOptimisticUpdate (Note 6)	optimistic	update	read committed

Note:
1. Read locks are held for the duration of the transaction.
2. The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
3. SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
4. A plain SELECT query is generated. No locks are held, but updates are permitted. Use cautiously. This intent enables execution without concurrency control.
5. Where supported by the backend, the generated SELECT query does not include `FOR UPDATE`; locks are escalated by the persistent store at storage time if updates were made. Otherwise, the same as wsPessimisticUpdate.
6. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

Be sure to review the rules for forming overqualified-update query predicates. Certain column types (for example, BLOB) are ineligible for inclusion in the overqualified-update query predicate and might affect your design.

Java Persistence API (JPA) architecture

Data persistence is the ability to maintain data between application executions. Persistence is vital to enterprise applications because of the required access to relational databases. Applications that are developed for this environment must manage persistence themselves or use third-party solutions to handle database updates and retrievals with persistence. The Java Persistence API (JPA) provides a mechanism for managing persistence and object-relational mapping and functions for the EJB 3.0 and EJB 3.1 specifications.

The JPA specification defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations. JPA is based on the Java programming model that applies to Java EE environments, but JPA can function within a Java SE environment for testing application functions.

JPA represents a simplification of the persistence programming model. The JPA specification explicitly defines the object-relational mapping, rather than relying on vendor-specific mapping implementations. JPA standardizes the important task of object-relational mapping by using annotations or XML to map objects into one or more tables of a database. To further simplify the persistence programming model:

- The EntityManager API can persist, update, retrieve, or remove objects from a database
- The EntityManager API and object-relational mapping metadata handle most of the database operations without requiring you to write JDBC or SQL code to maintain persistence
- JPA provides a query language, extending the independent EJB querying language (also known as JPQL), that you can use to retrieve objects without writing SQL queries specific to the database you are working with.

JPA is designed to operate both inside and outside of a Java Enterprise Edition (Java EE) container. When you run JPA inside a container, the applications can use the container to manage the persistence context. If there is no container to manage JPA, the application must handle the persistence context management itself. Applications that are designed for container-managed persistence do not require as much code implementation to handle persistence, but these applications cannot be used outside of a container. Applications that manage their own persistence can function in a container environment or a Java SE environment.

Java EE containers that support the EJB 3.x programming model must support a JPA implementation, also called a persistence provider. A JPA persistence provider uses the following elements to allow for easier persistence management in an EJB 3.x environment:

Persistence unit
Consists of the declarative metadata that describes the relationship of entity class objects to a relational database. The EntityManagerFactory uses this data to create a persistence context that can be accessed through the EntityManager.

EntityManagerFactory
Used to create an EntityManager for database interactions. The application server containers typically supply this function, but the EntityManagerFactory is required if you are using JPA application-managed persistence. An instance of an EntityManagerFactory represents a Persistence Context.

Persistence context
Defines the set of active instances that the application is manipulating currently. The persistence context can be created manually or through injection.

EntityManager
The resource manager that maintains the active collection of entity objects that are being used by the application. The EntityManager handles the database interaction and metadata for object-relational mappings. An instance of an EntityManager represents a Persistence Context. An application in a container can obtain the EntityManager through injection into the application or by

looking it up in the Java component name-space. If the application manages its persistence, the EntityManager is obtained from the EntityManagerFactory.

Attention: Injection of the EntityManager is only supported for the following artifacts:

- EJB 3.x session beans
- EJB 3.x message-driven beans
- Servlets, Servlet Filters, and Listeners
- JSP tag handlers which implement interfaces javax.servlet. jsp.tagext.Tag and javax.servlet.jsp.tagext.SimpleTag
- JavaServer Faces (JSF) managed beans
- the main class of the application client.

Entity objects

A simple Java class that represents a row in a database table in its simplest form. Entities objects can be concrete classes or abstract classes. They maintain states by using properties or fields.

For more information about persistence, see the section on Java Persistence API Architecture and the section on Persistence in the Apache OpenJPA User Guide. For more information and examples on specific elements of persistence, see the sections on the EntityManagerFactory, and the EntityManager in the Apache OpenJPA User Guide.

JPA for WebSphere Application Server

Java Persistence API (JPA) 2.0 for WebSphere Application Server is built on the Apache OpenJPA 2.x open source project.

Apache OpenJPA is a compliant implementation of the JPA specification. Using OpenJPA as a base implementation, WebSphere Application Server employs extensions to provide additional features and utilities for WebSphere Application Server customers. Because JPA for WebSphere Application Server is built from OpenJPA, all OpenJPA function, extensions, and configurations are unaffected by the WebSphere Application Server extensions. You do not need to make changes to OpenJPA applications to use these applications in WebSphere Application Server.

JPA for WebSphere Application Server provides more than compatibility with OpenJPA. JPA for WebSphere Application Server contains a set of tools for application development and deployment. Other features of JPA for WebSphere Application Server include support for DB2 Optim pureQuery Runtime, DB2 optimizations, JPA Access Intent, enhanced tracing capabilities, command scripts, and translated message files. The provider of JPA for WebSphere Application Server is `com.ibm.websphere.persistence.PersistenceProviderImpl`.

Apache OpenJPA supports the use of properties to configure the persistent environment. JPA for WebSphere Application Server properties can be specified with either the openjpa or wsjpa prefix. You can mix the openjpa and wsjpa prefixes as you wish for a common set of properties. Exceptions to the rule are wsjpa specific configuration properties, which use the wsjpa prefix. When a JPA for WebSphere Application Server-specific property is used with the openjpa prefix, a warning message is logged indicating that the offending property is treated as a wsjpa property. The reverse does not hold true for the openjpa prefix. In that case, the offending property is ignored.

wsjpaversion command

Use this command-line tool to find out information about the installed version of Java Persistence API (JPA) for WebSphere Application Server.

Run the JPA commands (.bat on Windows or .sh on UNIX) from the `<`*`profile_root`*`>/bin` directory, to make sure that you have the latest version of the commands for your release.

Syntax

The command syntax is as follows:

Usage

The version tool can be useful when debugging problems with JPA in applications and providing customer support teams with the information about the current JPA environment.

The command is run from the *<profile_root>* directory.

Examples

Find the version information of your JPA installation example output:

```
[root@atlanta bin]# ./wsjpaversion.sh
WSJPA 2.1.0-SNAPSHOT
version id: WSJPA-2.1.0-SNAPSHOT-r1119:2233
WebSphere JPA svn revision: 1119:2233

OpenJPA 2.1.0-SNAPSHOT
version id: openjpa-2.1.0-SNAPSHOT-r422266:1069208
Apache svn revision: 422266:1069208

os.name: Linux
os.version: 2.6.18-238.1.1.el5
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
 /root/tc/WASX/as/dev/JavaEE/j2ee.jar
 /root/tc/WASX/as/plugins/com.ibm.ws.jpa.jar
 /root/tc/WASX/as/plugins/com.ibm.ws.prereq.commons-collections.jar

/root/tc/WASX/as/profiles/AppSrv01/bin
[root@atlanta bin]#
```

On Windows operating systems, the output looks like the following:

```
D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin>wsjpaversion.bat
WSJPA 2.1.0-SNAPSHOT
version id: WSJPA-2.1.0-SNAPSHOT-r1119:2216
WebSphere JPA svn revision: 1119:2216
OpenJPA 2.1.0-SNAPSHOT
version id: openjpa-2.1.0-SNAPSHOT-r422266:1063829
Apache svn revision: 422266:1063829
os.name: Windows 7
os.version: 6.1
os.arch: amd64
java.version: 1.6.0
java.vendor: IBM Corporation
java.class.path:
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\dev\JavaEE\j2ee.jar
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\plugins\com.ibm.ws.jpa.jar
        D:\Users\user\WASV8\IBM\WebSphere\AppServer\plugins\com.ibm.ws.prereq.
commons-collections.jar
        .
        C:\Program Files (x86)\IBM\Java60\jre\lib\ext\QTJava.zip
user.dir: D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin
D:\Users\user\WASV8\IBM\WebSphere\AppServer\bin>
```

Examples

Find the version information of your JPA installation example output:

```
C:\was70-GM>profiles\a1002.07\bin\wsjpaversion.bat
WSJPA 2.0.0-SNAPSHOT
version id: WSJPA-2.0.0-SNAPSHOT-r1118:1843
revision: 1118:1843

OpenJPA 2.0.0-SNAPSHOT
version id: openjpa-2.0.0-SNAPSHOT-r422266:897308
Apache svn revision: 422266:897308

os.name: Windows XP
os.version: 5.1 build 2600 Service Pack 2
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
        C:\was70-GM\feature_packs\jpa\dev\JavaEE\j2ee.jar
        C:\was70-GM\feature_packs\jpa\plugins\com.ibm.ws.jpa.jar
        C:\was70-GM\plugins\com.ibm.ws.prereq.commons-collections.jar

user.dir: C:\was70-GM\plugins\com.ibm.ws.jpa.jar
```

wsjpa properties

The extension properties of Java Persistence API (JPA) for WebSphere Application Server can be specified with the openjpa or wsjpa prefix. This topic features the wsjpa properties.

wsjpa.AccessIntent

Use this property to define a TaskName that in the `persistence.xml` file using the wsjpa.AccessIntent property name in a persistence unit. The property value is a list of TaskNames, entity types and access intent definitions.

For more information and examples on how the wsjpa.AccessIntent property is used, see the topic Specifying TaskName in a JPA persistence unit.

wsjpa.jdbc.Schema

Specifies the schema name in a DB2 package collection when using multiple DB2 package collections.

For more information about using the wsjpa.jdbc.Schema property see the topic, Configuring pureQuery to use multiple DB2 package collections.

wsjpa.jdbc.CollectionId

Specifies the collection Id name in a DB2 package collection when using multiple DB2 package collections.

For more information about using the wsjpa.jdbc.CollectionId property see the topics, Configuring pureQuery to use multiple DB2 package collections and Configuring data source JDBC providers to use pureQuery in a Java SE environment.

Transaction support in WebSphere Application Server

Support for transactions is provided by the transaction service within WebSphere Application Server. The way that applications use transactions depends on the type of application component.

A transaction is unit of activity, within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, during the processing of an SQL COMMIT statement, the database manager atomically commits multiple SQL statements to a relational database. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a *logical unit of work* (LUW). If a transaction involves multiple resource managers, for example multiple database managers, an external transaction manager is required to coordinate the individual resource managers. A transaction that spans multiple resource managers is referred to as a *global transaction*. WebSphere Application Server is a transaction manager that can coordinate global transactions, can be a participant in a received global transaction, and can also provide an environment in which resource manager local transactions can run.

The way that applications use transactions depends on the type of application component, as follows:
- A session bean can use either container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (component-managed transactions where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) and application client components use component-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface, and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service (OTS) protocol or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through Java EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with databases, JMS queues, and JCA connectors through their *local transaction* support, when you do not require distributed transaction coordination.

IBM i Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances, you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to complete any updates, the one-phase commit resource does not have to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one or more two-phase commit resource providers and where *last participant support* is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the Java EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:
- Enterprise Information Systems that are accessed through a resource adapter, as described in the Java EE Connector Architecture.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. Resource adapter components of the Java EE connector architecture that include support for local transactions provide a LocalTransaction interface. The LocalTransaction interface enables applications to request that the resource adapter commits or rolls back RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, an external transaction manager is needed to coordinate the updates to all the resource managers in a global transaction.

Global transaction support is available to web and enterprise bean components and, with some limitations, to application client components. Enterprise bean components can be subdivided into two categories: beans that use container-managed transactions (CMT) and beans that use bean-managed transactions (BMT).

Only BMT enterprise beans, application client components, and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. To obtain the UserTransaction interface, use a Java Naming and Directory Interface (JNDI) lookup of java:comp/UserTransaction, or use the getUserTransaction method from the SessionContext object.

The UserTransaction interface is not available to CMT enterprise beans. If CMT enterprise beans attempt to obtain this interface, an exception is thrown, in accordance with the Enterprise JavaBeans (EJB) specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location that is appropriate for the EJB version.

WebSphere Application Server Version 4 and later releases bind the UserTransaction interface at the JNDI location that is specified in the EJB Version 1.1 specification. This location is java:comp/UserTransaction.

A web component or enterprise bean (CMT or BMT) can use additional interfaces that provide JTA support. These interfaces provide the transaction identity and a mechanism to receive notification of transaction completion. The interfaces include the TransactionSynchronizationRegistry interface, the ExtendedJTATransaction interface, and the UOWSynchronizationRegistry interface.

Local transaction containment

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

Unspecified transaction context is defined in the Enterprise JavaBeans specification, Version 2.0 and later. For example, see the specification for this technology.

An LTC is a bounded unit-of-work scope, within which zero or more resource manager local transactions (RMLT) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. By default, an LTC is local to a bean instance; it is not shared across beans, even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on an enterprise application component, such as an enterprise bean or servlet, whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example, at the end of the method dispatch. There is no programmatic interface to the LTC support; LTCs are managed exclusively by the container. The application deployer configures LTCs on individual application components, either web application or EJB, by using transaction attributes in the application deployment descriptor.

A local transaction containment (LTC) might be configured as part of an application component's deployment descriptor to be shareable across multiple application components, including web application components and enterprise beans that use container-managed transactions, so that those components can share connections without using a global transaction. Sharing a single resource manager between application components improves performance, increases scalability, and reduces lock contention for resources.

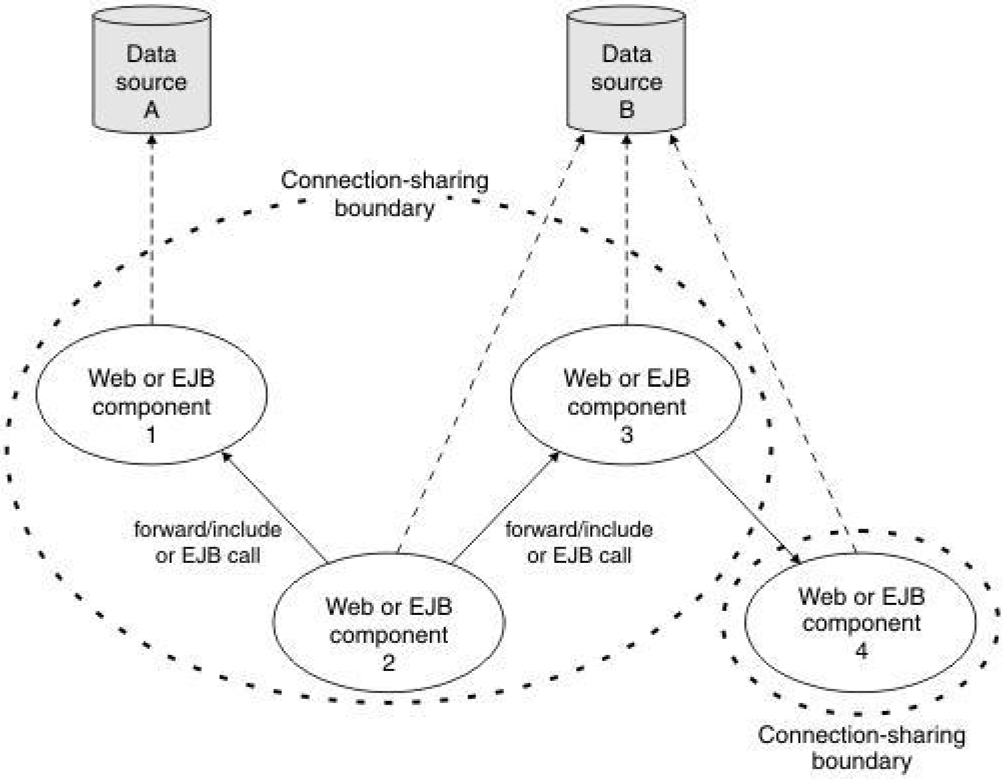
LTCs can be shared across multiple components, including web application components and enterprise beans that use container-managed transactions. This sharing is useful in situations such as frequent use of web component `include()` calls, where a thread can have several connections blocked by LTCs in different web modules. In this situation, the application might encounter code deadlocks under load, when threads start to wait for themselves to free connections. To overcome this issue without using a global transaction, specify that application components can share LTCs by setting the Shareable attribute in the deployment descriptor of each component. You must use a deployment descriptor; you cannot specify this attribute if annotation has been used.

When you set the Shareable attribute, the extended deployment descriptor XML file includes the following line of code:

```
<local-transaction boundary="BEAN_METHOD" resolver="CONTAINER_AT_BOUNDARY"
unresolved-action="COMMIT" shareable="true"/>
```

To obtain the full benefits of a shared LTC, also ensure that the resource reference for each component defaults to shareable connections.

In the following diagram, components 1, 2 and 3 are deployed with the Shareable attribute and component 4 is not. If components 2 and 3 both obtain connections to data source B, and their resource references for data source B default to shareable connections, they share the connection, but component 4 does not.



Applications that use shareable LTCs cannot explicitly commit or roll back resource manager connections that are used in a shareable LTC. Although, they can use connections that have an autoCommit capability. This ensures correct encapsulation of connection usage by each component and protects one component from having to make any assumptions about the behavior of other components that share the connection.

If an application starts any non-autocommit work in an LTC for which the Resolver attribute is set to `Application` and the Shareable attribute is set to `true`, an exception occurs at run time. For example, on a JDBC connection, non-autocommit work is work that the application performs after using the setAutoCommit(false) method to disable the autocommit option on the connection. Enterprise beans that use bean managed transactions (BMT) cannot be assembled with the Shareable attribute set on the LTC configuration.

IBM i A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for enterprise application components at J2EE 1.3 or later. The only exceptions to this are the following items:

- Application component dispatch occurs without container interposition, for example, for a stateless session bean create method or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 bean-managed transaction (BMT) enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that exists longer than the enterprise bean method in which it is started, as described in the topic about ActivitySessions and transaction contexts.

Local transaction containment

IBM WebSphere Application Server supports local transaction containment (LTC), which you can configure using local transaction extended deployment descriptors. LTC support provides certain advantages to application programmers. Use the scenarios provided, and the list of points to consider, to help you decide the best way to configure transaction support for local transactions.

The following sections describe the advantages that LTC support provides, and how to set the local transaction extended deployment descriptors in each situation.

You can develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not have to use global transactions, it is often more efficient to deploy the bean with the deployment descriptor for the container transaction type set to NotSupported instead of Required.

With the extended local transaction support of the application server, applications can perform the same business logic in an unspecific transaction context as they can in a global transaction. An enterprise bean, for example, runs in an unspecified transaction context if it is deployed with a container transaction type of NotSupported or Never.

The extended local transaction support provides a container-managed, implicit local transaction boundary, within which the container commits application updates and cleans up their connections. You can design applications with more independence from deployment concerns. This makes using a container transaction type of Supports much simpler, for example, when the business logic might be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage, regardless of whether the application runs in a transaction. The application can depend on the close action behaving in the same way in all situations, that is, the close action does not cause a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios, running business logic in a Transaction policy of NotSupported performs better than in a policy of Required. This benefit is applied through setting the deployment descriptor, in the Local Transactions section, of the Resolver attribute to ContainerAtBoundary. With this setting, application interactions with resource providers, such as databases, are managed within implicit resource manager local transactions (RMLT) that the container both starts and ends. The container commits RMLTs at the containment boundary that is specified by the Boundary attribute in the Local Transactions section; for example, at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

You can use local transactions in a managed environment that guarantees cleanup.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default setting of Application for the Resolver extended deployment descriptor in the Local Transactions section. In this situation, the container ensures connection cleanup at the boundary of the local transaction context.

Java platform for enterprise applications specifications that describe application use of local transactions do so in the manner provided by the default settings of Application for the Resolver extended deployment descriptor, and Rollback for the Unresolved action extended deployment descriptor, in the Local Transactions section. When the Unresolved action extended deployment descriptor in the Local Transactions section is set to Commit, the container commits any RMLTs that the application starts but that do not complete when the local transaction containment ends (for example, when the method ends). This usage applies to both servlets and enterprise beans.

IBM i **You can extend the duration of a local transaction beyond the duration of an EJB component method.**

The Enterprise JavaBeans (EJB) specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can use the Boundary extended deployment setting in the Local Transactions section to give the following advantages:

- Significantly extend the use cases of RMLTs.

- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which a client can control. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is applied through the Boundary extended deployment setting, in the Local transactions section, of ActivitySession. Such extended RMLTs can remain under the control of the application, or be managed by the container, depending on the setting of the Resolver deployment descriptor in the Local Transactions section.

You can coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can use ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans in that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

You can use shareable LTCs to reduce the number of connections you require.

Application components can share LTCs. If components obtain connections to the same resource manager, they can share that connection if they run under the same global transaction or shareable LTC. To configure two components to run under the same shareable LTC, set the Shareable attribute of the Local Transactions section in the deployment descriptor of each component. Make sure that the resource reference in the deployment descriptor for each component uses the default value of Shareable for the res-sharing-scope element, if this element is specified. A shareable LTC can reduce the numbers of RMLTs an application uses. For example, an application that makes frequent use of web module include calls can share resource manager connections between those web modules, exploiting either shareable LTCs, or a global transaction, reducing lock contention for resources.

Examples of local transaction support configurations

The following list gives scenarios that use local transactions, and points to consider when deciding the best way to configure the transaction support for an application.

- You want to start and end global transactions explicitly in the application (bean-managed transaction session beans and servlets only).

  For a session bean, set the Transaction type to Bean (to use bean-managed transactions) in the deployment descriptor of the component. You do not have to do this for servlets.
- IBM i You want to access only one XA or non-XA resource in a method.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to Supports.
- You want to access several XA resources atomically across one or more bean methods.

  In the deployment descriptor of the component, in the Container Transactions section, set the container transaction type to Required, RequiresNew, or Mandatory.
- IBM i You want to access several non-XA resources in a method without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to NotSupported.
- You want to access several non-XA resources in a method and want to manage them independently.

In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Unresolved action attribute to Rollback. In the Container Transactions section, set the container transaction type to NotSupported.

- IBM i You want to access one or more non-XA resources across multiple EJB method calls without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.
- IBM i You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.

Local and global transactions

Applications use resources, such as Java Database Connectivity (JDBC) data sources or connection factories, that are configured through the Resources view of the administrative console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider.

For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLT), but an XA data source can support two-phase commit coordination, as well as local transactions.

IBM i If an application uses two or more resource providers that support only RMLTs, atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application must use resources that support XA coordination and must access those resources in a global transaction.

If an application uses only one RMLT, atomic behavior can be guaranteed by the resource manager, which can be accessed in a local transaction containment (LTC) context.

An application can also access a single resource manager in a global transaction context, even if that resource manager does not support the XA coordination. An application can do this because the application server performs an “only resource optimization” and interacts with the resource manager in a RMLT. In a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but not both. An instance of an enterprise bean can change from running in one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions

Application clients can, within certain limits, support the use of transactions.

Application clients running in an enterprise application client container can explicitly demarcate transaction boundaries, as described in the topic about using component-managed transactions. Application clients cannot perform, directly in the client container, transactional work in the context of any global transaction that they start, because the client container is not a recoverable process.

Application clients can make requests to remote objects, such as enterprise beans, in the context of a client-initiated transaction. Any transactional work performed in a remote, recoverable, server process is coordinated as part of the client-initiated transaction. The transaction coordinator is created on the first server process to which the client-initiated transaction is propagated.

A client can begin a transaction, then, for example, access a JDBC data source directly in the client process. In such cases, any work performed through the JDBC provider is not coordinated as part of the global transaction. Instead, the work runs under a resource manager local transaction. The client container process is non-recoverable and contains no transaction coordinator with which a resource manager can be enlisted.

A client can begin a transaction, then call a remote application component such as an enterprise bean. In such cases, the client-initiated transaction context is implicitly propagated to the remote application server, where a transaction coordinator is created. Any resource managers accessed on the recoverable application server (or any other application server hosting application components invoked by the client) are enlisted in the global transaction.

Client application components must be aware that locally-accessed resource managers are not coordinated by client-initiated transactions. Client applications acknowledge this through a deployment option that enables access to the UserTransaction interface in the client container. By default, access to the UserTransaction interface in the client container is not enabled. To enable UserTransaction demarcation for an application client component, set the “Allow JTA Demarcation” extension property in the client deployment descriptor. For information about editing the client deployment descriptor, refer to the Rational Application Developer information.

Commit priority for transactional resources

You can specify the order in which transactional resources are processed during two-phase commit processing.

If you control the order in which transactional resources are processed during two-phase commit processing, there are two main benefits:

- One-phase commit optimization occurs more often.
- Potential problems caused by transaction isolation are resolved.

To control the order in which transactional resources are processed during two-phase commit processing, you specify the commit priority of a resource by setting the commit priority attribute on a resource reference. The larger the commit priority, the earlier the resource is processed. For example, if a resource has a commit priority of 10, it is processed before a resource with a commit priority of 1. The commit priority value is of type int and can be between -2147483648 and 2147483647.

If you do not specify a commit priority value, a default value of zero is assigned to the resource and is used when ordering resources at run time. If two or more resources are configured with the same priority, including the default priority, they are processed in an unspecified order with respect to each other.

You can specify the commit priority attribute on a resource reference by using Rational Application Developer tools. For detailed information, see the Rational Application Developer information center. The application component must have a deployment descriptor; you cannot specify this attribute if annotation has been used.

One-phase commit optimization

In a transaction with a two-phase commit, if every resource except the last one enlisted in the transaction votes read-only, indicating that those resources are not interested in the outcome of the transaction, a one-phase commit can occur. This means that the transaction service does not have to store resource and transaction information that it would need to roll back a two-phase commit, and therefore performance is improved.

You can control the order in which transactional resources are processed during two-phase commit, so you can process the resources that are most likely to vote read-only first. Therefore, you increase the chance that a one-phase commit might occur.

Typically, for a given transactional resource, you know the work that is performed at run time, so if you can control the order in which the resources in a transaction are processed, you can increase the likelihood of a one-phase commit optimization occurring.

Transaction isolation

When resources are involved in a global transaction, updates that are made as part of a transaction are not visible outside the transaction until the transaction commits, that is, those resources are isolated. This isolation can cause problems with other application components that act on the updates after they are committed. For example, further processing can fail, or can fail intermittently, because updates are order and time dependent. This problem does not occur with service integration bus messaging work in WebSphere Application Server, but can be a problem for other messaging providers, for example WebSphere MQ.

If you specify the order in which transactional resources are committed, problems caused by isolation are resolved for all transactional systems, not just messaging providers and service integration bus in particular.

The following example describes how problems might occur when you cannot specify the order in which transactional resources are committed. An application updates a row in a database table, then sends a JMS message that triggers additional processing of the row. Both of these actions are performed in the same global transaction, so they are isolated until their respective resources are committed. If the update to the row is committed before the message is sent, the processing that is triggered by the message can access the updated row and process it. If the action to send the message is committed first, this action might trigger the additional processing of the row before the database has committed the update to the row. In this situation, the updated row is still isolated and is not visible, so the additional processing of the row fails.

This problem can be more complicated because it is ordering and timing dependent. If the database is committed first, the problem does not occur. If the action to send the message is committed first, the problem might occur, but it depends whether the database work is committed before the message triggers the further processing of the row. Therefore, the problem can be intermittent, so it is harder to identify its cause.

Restrictions with earlier versions of WebSphere Application Server

If you specify the commit priority of a resource, that is, specify any value other than the default value 0, the commit priority is added to the partner log in a recoverable unit section. This section in the log file is recognized in WebSphere Application Server Version 7.0 or later, but not in earlier versions of the application server.

Therefore, if an application uses the commit priority attribute, you cannot install that application into a mixed-version cluster where one or more servers in the cluster are at versions of WebSphere Application Server that are earlier than Version 7.0.

Also, if an application that uses the commit priority attribute is installed in a cluster, you cannot subsequently add a server to that cluster if the server is at a version of WebSphere Application Server that is earlier than Version 7.0.

For general information about different versions of the product, see the topic “Overview of migration, coexistence, and interoperability”.

Transaction compensation and business activity support

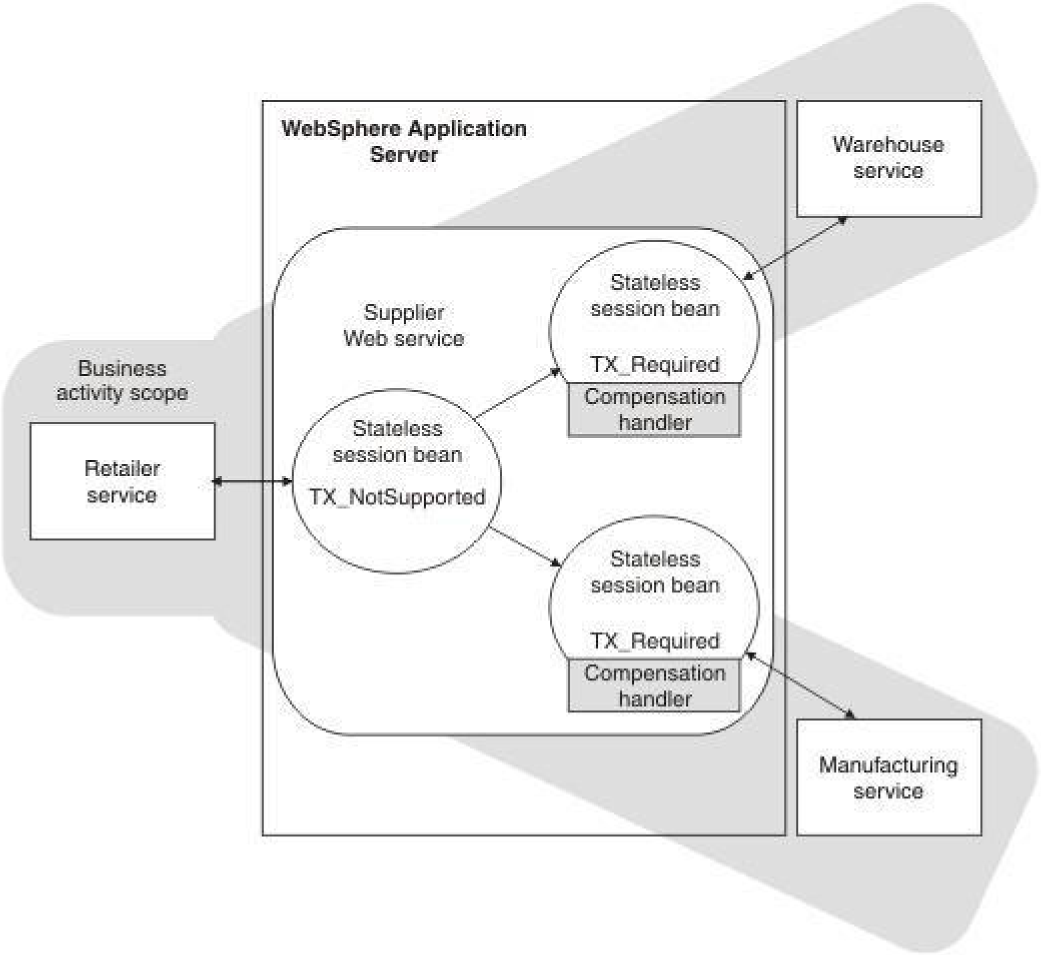
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an email can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Serverenvironments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that undertake work that can be compensated. These other session beans each have a *compensation handler*; a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an email.



Application design

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work by using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functions. Enterprise beans that exploit business activity scopes can offer web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in web service messages by using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

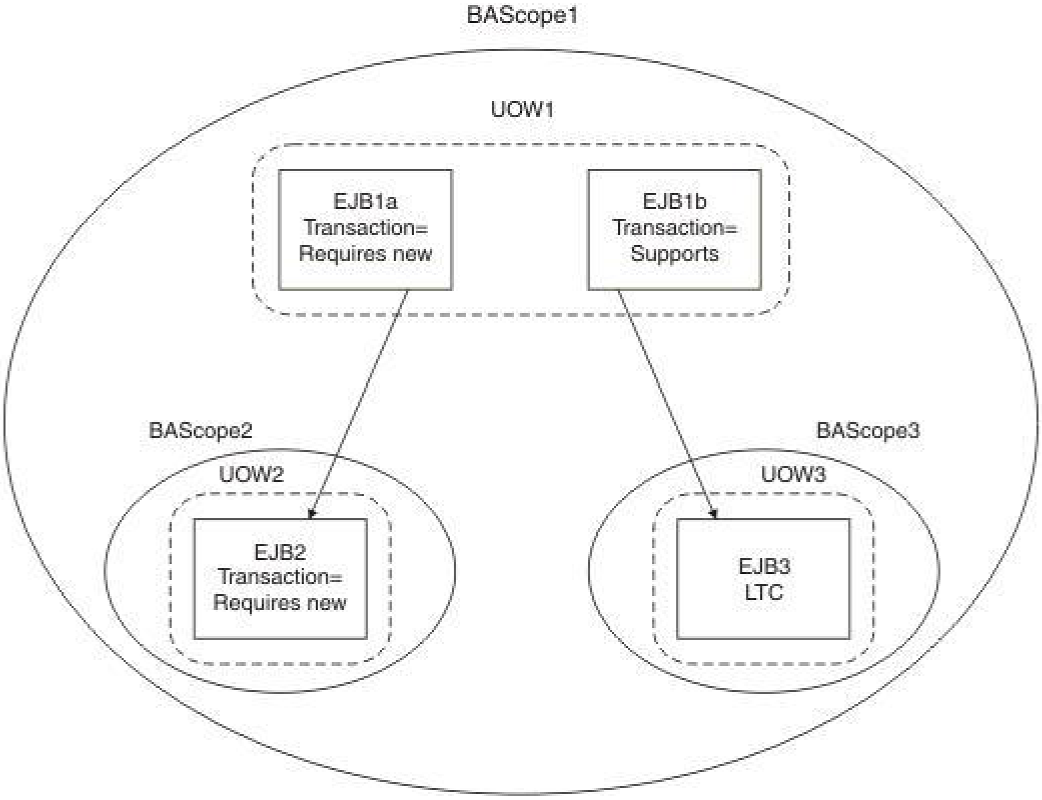
Business activity scopes

The scope of a business activity is that of a main WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing main UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any main UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.


BAScope1
UOW1
EJB1a Transaction= Requires new
EJB1b Transaction= Supports
BAScope2
UOW2
EJB2 Transaction= Requires new
BAScope3
UOW3
EJB3 LTC

Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by trying the called component again or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 21. Compensation behavior for a single business activity scope. The table lists the possible combinations of success and failure for the inner and outer business activity scopes, and the compensation behavior associated with each combination.

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see the topic about the business activity API.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight by using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration

If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope

Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the

received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method, then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For details, see the topic about creating an application that uses the WS-BA support.

JTA support

Java Transaction API (JTA) support provides application programming interfaces (APIs) in addition to the UserTransaction interface that is defined in the JTA 1.1 specification.

These interfaces include the TransactionSynchronizationRegistry interface, which is defined in the JTA 1.1 specification, and the following API extensions:
- SynchronizationCallback interface
- ExtendedJTATransaction interface
- UOWSynchronizationRegistry interface
- UOWManager interface

The APIs provide the following functions:
- Access to global and local transaction identifiers associated with the thread.

  The global identifier is based on the transaction identifier in the CosTransactions::PropagationContext: object and the local identifier identifies the transaction uniquely in the local Java virtual machine (JVM).
- A transaction synchronization callback that any enterprise application component can use to register an interest in transaction completion.

  Advanced applications can use this callback to flush updates before transaction completion and clear up state after transaction completion. Java EE (and related) specifications position this function typically as the domain of the enterprise application containers.

  Components such as persistence managers, resource adapters, enterprise beans, and web application components can register with a JTA transaction.

The following information is an overview of the interfaces that the JTA support provides. For more detailed information, see the generated API documentation.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run on a Java platform for enterprise applications server, there is no specific enterprise application component active when this object is called. So, the object has limited direct access to any enterprise application resources. Specifically, the object has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to an enterprise application component (for example, a stateless session bean) that it delegates to. The object would then have all the usual access to enterprise application resources. For example, you might use the object to acquire a Java Database Connectivity (JDBC) connection and flush updates to a database during the beforeCompletion method.

ExtendedJTATransaction interface

This interface is a WebSphere programming model extension to the Java EE JTA support. An object implementing this interface is bound, by enterprise application containers in WebSphere Application Server

that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an Enterprise JavaBeans (EJB) container, is not restricted to component-managed transactions.

An application uses a Java Naming and Directory Interface (JNDI) lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which the application uses as shown in the following example:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallback objects. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server, whether the transaction is started locally or imported
- At the end of the transaction for which the callback was registered

Note: In this release, the registerSynchronizationCallbackForCurrentTran method is deprecated. Use the registerInterposedSynchronization method of the TransactionSynchronizationRegistry interface instead.

TransactionSynchronizationRegistry interface

This interface is defined in the JTA 1.1 specification. System-level application components, such as persistence managers, resource adapters, enterprise beans, and web application components, can use this interface to register with a JTA transaction. Then, for example, the component can flush a cache when a transaction completes.

To obtain the TransactionSynchronizationRegistry interface, use a JNDI lookup of java:comp/TransactionSynchronizationRegistry.

Note: Use the registerInterposedSynchronization method to register a synchronization instance, rather than the registerSynchronizationCallbackForCurrentTran method of the ExtendedJTATransaction interface, which is deprecated in this release.

UOWSynchronizationRegistry interface

This interface provides the same functions as the TransactionSynchronizationRegistry interface, but applies to all types of units of work (UOWs) that WebSphere Application Server supports:

- JTA transactions
- local transaction containments (LTCs)
- ActivitySession contexts

System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface to register with a JTA transaction. The component can do the following:

- Register synchronization objects with special ordering semantics.
- Associate resource objects with the UOW.
- Get the context of the current UOW.
- Get the current UOW status.
- Mark the current UOW for rollback.

To obtain the UOWSynchronizationRegistry interface, use a JNDI lookup of java:comp/websphere/UOWSynchronizationRegistry. This interface is available only in a server environment.

The following example registers an interposed synchronization with the current UOW:

```
// Retrieve an instance of the UOWSynchronizationRegistry interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWSynchronizationRegistry uowSyncRegistry =
  (UOWSynchronizationRegistry)initialContext.lookup("java:comp/websphere/UOWSynchronizationRegistry");

// Instantiate a class that implements the javax.transaction.Synchronization interface
final Synchronization sync = new SynchronizationImpl();

// Register the Synchronization object with the current UOW.
uowSynchronizationRegistry.registerInterposedSynchronization(sync);
```

UOWManager interface

The UOWManager interface is equivalent to the JTA TransactionManager interface, which defines the methods that allow an application server to manage transaction boundaries. Applications can use the UOWManager interface to manipulate UOW contexts in the product. The UOWManager interface applies to all types of UOWs that WebSphere Application Server supports; that is, JTA transactions, local transaction containments (LTCs), and ActivitySession contexts. Application code can run in a particular type of UOW without needing to use an appropriately configured enterprise bean. Typically, the logic that is performed in the scope of the UOW is encapsulated in an anonymous inner class. System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface.

WebSphere Application Server does not provide a TransactionManager interface in the API or the system programming interface (SPI). The UOWManager interface provides equivalent functions, but WebSphere Application Server maintains control and integrity of the UOW contexts.

To obtain the UOWManager interface in a container-managed environment, use a JNDI lookup of java:comp/websphere/UOWManager. To obtain the UOWManager interface outside a container-managed environment, use the UOWManagerFactory class. This interface is available only in a server environment.

You can use the UOWManager interface to migrate a web application to use web components rather than enterprise beans, but maintain control over the UOWs. For example, a web application currently uses the UserTransaction interface to begin a global transaction, makes a call to a method on a session enterprise bean that is configured as not supported to undertake some non-transactional work, and then completes the global transaction. You can move the logic that is encapsulated in the session EJB method to the run method of a UOWAction implementation. Then, you replace the code in the web component that calls the session enterprise bean with a call to the runUnderUOW method of a UOWManager interface to request that this logic is run in a local transaction. In this way, you maintain the same level of control over the UOWs as you had with the original application.

The following example performs some transactional work in the scope of a new global transaction. The transactional work is performed in an anonymous inner-class that implements the run method of the UOWAction interface. Any checked exceptions that the run method creates do not affect the outcome of the transaction.

```
// Retrieve an instance of the UOWManager interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWManager uowManager = (UOWManager)initialContext.lookup("java:comp/websphere/UOWManager");

try
{
 // Invoke the runUnderUOW method, indicating that the logic should be run in a global
 // transaction, and that any existing global transaction should not be joined, that is,
 // the work must be performed in the scope of a new global transaction.
 uowManager.runUnderUOW(UOWSynchronizationRegistry.UOW_TYPE_GLOBAL_TRANSACTION, false, new UOWAction()
 {
   public void run() throws Exception
   {
```

```
   // Perform transactional work here.
   }
 });
}

catch (UOWActionException uowae)
{
 // Transactional work resulted in a checked exception being thrown.
}

catch (UOWException uowe)
{
 // The completion of the UOW failed unexpectedly. Use the getCause method of the
 // UOWException to retrieve the cause of the failure.
}
```

Chapter 11. IBM WebSphere Application Server Developer Tools for Eclipse overviewVersion 8.5

IBM WebSphere Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java EE, OSGi, Web 2.0, and mobile applications to WebSphere Application Server.

When combined with WebSphere Application Server V8.5 Liberty Profile, WebSphere Application Server Developer Tools for Eclipse provides a fast and lightweight environment for the rapid development and unit testing of web, Web 2.0, mobile, and OSGi applications.

Supported programming models

The following table lists the programming models available in IBM WebSphere Application Server Developer Tools for Eclipse and indicates the application servers that they are supported on.

Programming model	WebSphere Application Server V8.5 Liberty Profile	WebSphere Application Server V8.5	WebSphere Application Server V8.0	WebSphere Application Server V7.0
Develop Java EE applications	Yes **Note:** EJB is not supported.	Yes	Yes	Yes
Develop JAX-RS applications	Yes	Yes	Yes	Yes
Develop JAX-WS applications	No	Yes	Yes	Yes
Develop Web 2.0 and mobile web applications	Yes	Yes	Yes	Yes
Develop OSGi applications	Yes **Note:** The following technologies are supported: • Web 2.5 • Web 3.0 • JPA	Yes **Note:** The following technologies are supported: • Web 2.5 • Web 3.0 • JPA • JAX-RS • JSF • EJB	Yes **Note:** The following technologies are supported: • Web 2.5 • Web 3.0 • JPA • JAX-RS • JSF	No

Advanced support

The following features are supported on all WebSphere Application Servers, except the Liberty profile:

- Migrate the server when importing applications targeted to an invalid server.
- Start and stop a remote server.
- Run and debug administrative script files on the server.

Note: The workbench does not support WebSphere Application Server V7.0 feature packs and managed (federated) WebSphere Application Server Network Deployment environments.

Getting started

To get started, see http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html

Visit the WASdev community site to:

- Participate in or post questions in the forum.
- Learn more about the workbench from the collection of articles and sample code.
- Get community news through blog entries.
- Help improve the workbench by reporting defects and requesting feature enhancements.

Getting help

WebSphere Application Server Developer Tools for Eclipse provides you with access to the Rational Application Developer documentation.

Note: Some documented features are available only with the full Rational Application Developer for WebSphere Software product.

For details about how to integrate the workbench with the server, see Testing and publishing on a server

Additional features available with Rational Application Developer for WebSphere Software

For enterprise-level development, consider using Rational Application Developer as your integrated development environment. Rational Application Developer provides a complete environment for Java, Java EE, web, web services, SOA, OSGi, Web 2.0, mobile, and portal designers and developers, including the server development tools. It also provides integration and support for the following application servers and associated feature packs:

- WebSphere Application Server V8.5 Liberty Profile
- WebSphere Application Server V8.5
- WebSphere Application Server V8.0
- WebSphere Application Server V7.0

Rational Application Developer accelerates development and unit test to ensure delivery of higher quality applications. It offers proven tools for teams using emerging trends of Web 2.0, SOA, OSGi, and Cloud computing that help accelerate adoption and software delivery.

For more information about this product, see Rational Application Developer for WebSphere Software or IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5 overview.

Chapter 12. The Liberty profile

The Liberty profile is a highly composable, fast to start, dynamic application server runtime environment.

You can install the server as described in Installing the Liberty profile. Because the Liberty profile does not include a Java runtime environment (JRE), you have to install a JRE provided by either IBM or Oracle beforehand. For more information about supported Java environments, and where to get them, see Minimum supported Java levels.

This server supports two models of application deployment:

- Deploy an application by dropping it into the “dropins” directory.
- Deploy an application by adding it to the server configuration.

The Liberty profile supports a subset of the following parts of the full WebSphere Application Server programming model:

- Web applications
- OSGi applications
- Java Persistence API (JPA)

Associated services such as transactions and security are only supported as far as is required by these application types and by JPA.

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server. The Liberty profile includes the following main features:

- Bean validation
- Blueprint
- Java API for RESTful Web Services
- Java Database Connectivity (JDBC)
- Java Naming and Directory Interface
- Java Persistence API (JPA)
- Java Server Faces (JSF)
- Java Server Pages (JSP)
- Local connector (for Java Management Extensions (JMX) clients)
- Monitoring
- OSGi JPA (JPA support for OSGi applications)
- Remote connector (for JMX clients)
- Secure Sockets Layer (SSL)
- Security
- Servlet
- Session persistence
- Transaction
- Web application bundle (WAB)

On distributed platforms, the Liberty profile provides both a development and an operations environment. On the Mac, it provides a development environment.

Liberty profile: Architecture

The liberty profile is a highly composable, dynamic runtime. OSGi services are used to manage component lifecycles, and the injection of dependencies and configuration. The server process comprises a single JVM, the Liberty kernel, and any number of optional features. The feature code and most of the kernel code runs as OSGi bundles within an OSGi framework. Features provide the programming models and services required by applications.

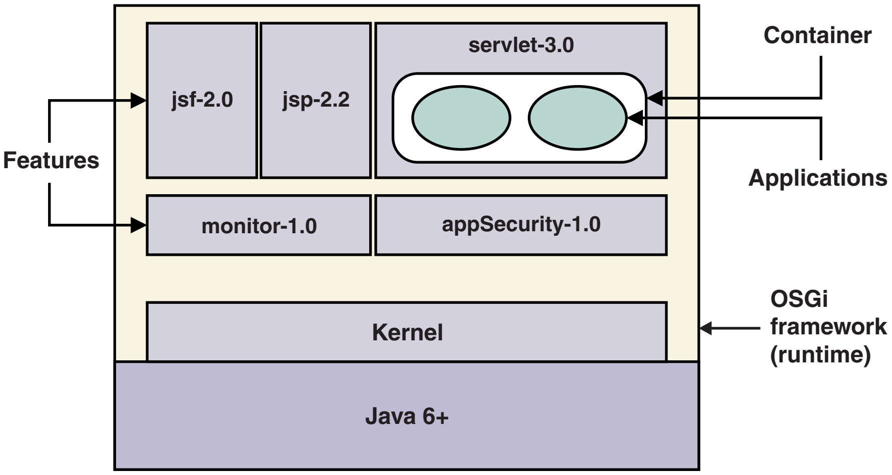


Figure 15. Liberty profile architecture

The kernel launcher bootstraps the system and starts the OSGi framework. The configuration is parsed, then the configured features are loaded by the feature manager. The kernel makes extensive use of OSGi services to provide a highly dynamic runtime. System configuration is managed by the OSGi Configuration Admin service, and an OSGi Declarative Services component is used to manage the lifecycle of system services. Application and configuration file changes are detected by the file monitor service, and the logging service writes messages and debug information to the local file system.

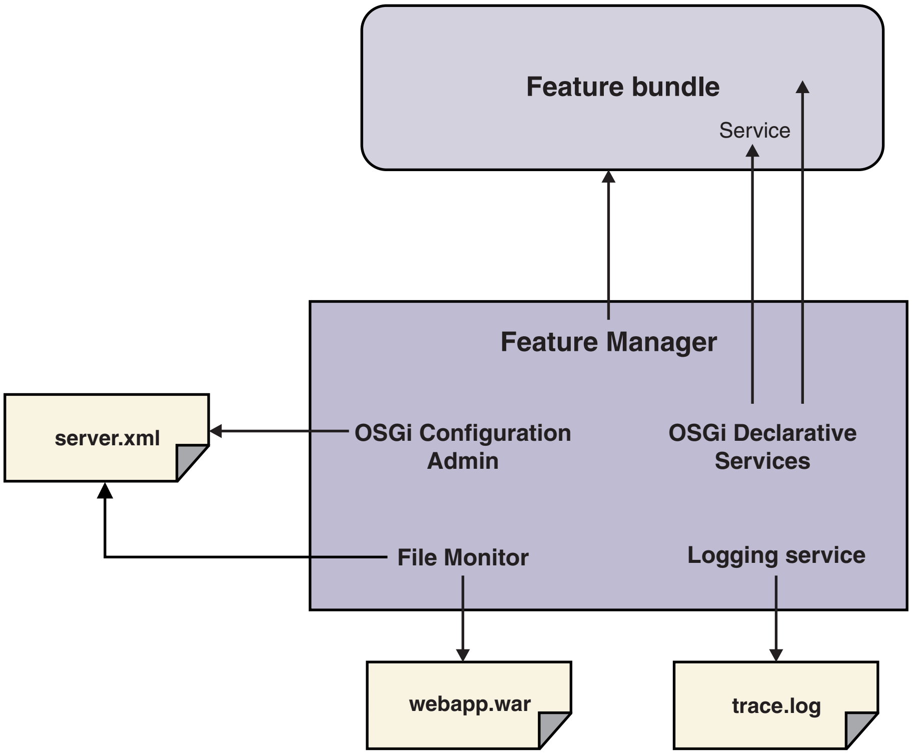


Figure 16. Liberty profile kernel

Features are specified in the system configuration files (that is, the `server.xml` file and any other included files). The server configuration files are used to populate the OSGi Configuration Admin service, which injects the feature configuration into the feature manager service. The feature manager maps each feature name to a list of bundles that provide the feature. The bundles are installed into the OSGi framework and started. The feature manager responds to configuration changes by dynamically adding and removing features while the server is running.

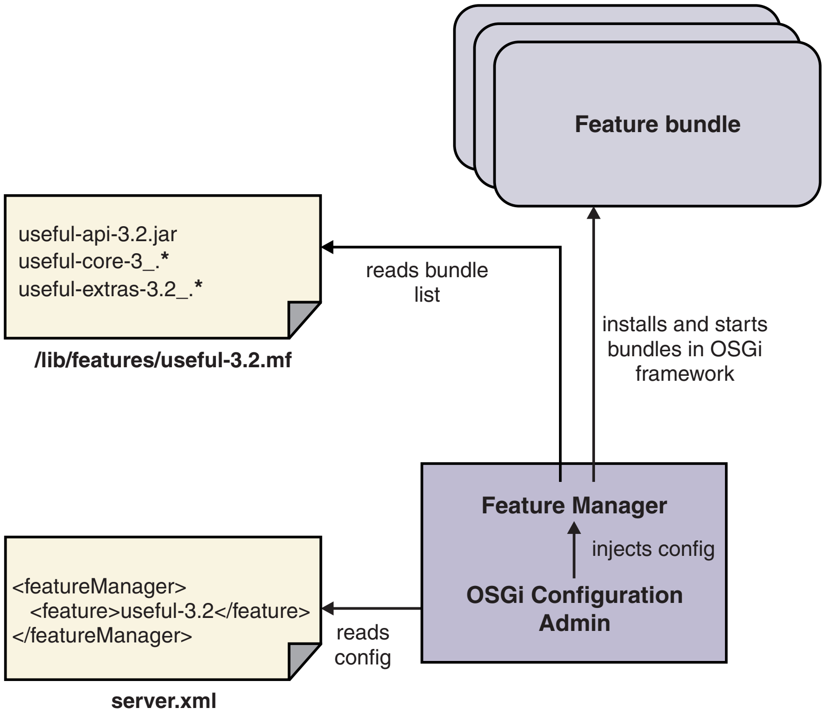


Figure 17. Feature management

Runtime services provide their configuration defaults so that the configuration you need to specify is kept to a minimum. You specify the features you need, along with any additions or overrides to the system defaults, in a `server.xml` file. You might choose to structure your configuration into a number of separate files, linked to the parent `server.xml` file using an “include” syntax. At server startup, or when the user configuration files are changed, the kernel configuration management parses your configuration and applies it over the system defaults. The set of configuration properties belonging to each service is injected into the service each time the configuration is updated.

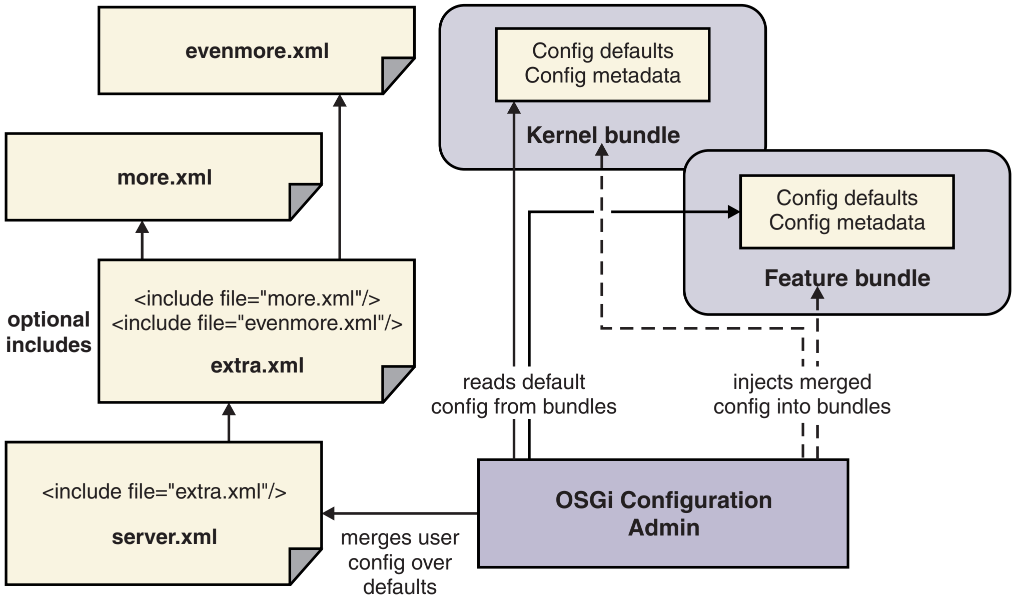

evenmore.xml
Config defaults
Config metadata
Kernel bundle
more.xml
Config defaults
Config metadata
Feature bundle
optional includes
<include file="more.xml"/>
<include file="evenmore.xml"/>
extra.xml
reads default config from bundles
injects merged config into bundles
<include file="extra.xml"/>
server.xml
merges user config over defaults
OSGi Configuration Admin

Figure 18. Configuration management

The OSGi Declarative Services component is used so that function can be decomposed into discrete services, which are activated only when needed. This helps the runtime to be “late and lazy”, keeping the footprint small and the startup fast. Declared services are added to the OSGi service registry, and dependencies between services can be resolved without loading implementation classes. Service activation can be delayed until a service is actually used: when the service reference is resolved. Configuration for each service is injected as the service is activated, and is reinjected if the configuration is later modified.

Programming model support

This set of tables shows the extent to which each of the major server profiles supports the full WebSphere Application Server programming model.

Java EE 6 technologies

Table 22. Java EE 6 support by profile.

A list of Java EE technologies, subdivided into sections for web services, web applications, enterprise applications, management and security, and Java EE-related specifications in Java SE. For each technology there is a specification reference, and an indication of whether the technology is supported by the full profile, and by the Liberty profile.

Technology	Specification reference	Full profile	Liberty profile
Java Platform, Enterprise Edition 6 (Java EE 6)	JSR 316	⌂	
Web services technologies			
Java API for RESTful Web Services (JAX-RS) 1.1	JSR 311	⌂	⌂
Implementing Enterprise Web Services 1.3	JSR 109	⌂	
Java API for XML-Based Web Services (JAX-WS) 2.2	JSR 224	⌂	
Java Architecture for XML Binding (JAXB) 2.2	JSR 222	⌂	⌂
Web Services Metadata for the Java Platform	JSR 181	⌂	
Java API for XML-Based RPC (JAX-RPC) 1.1	JSR 101	⌂	
Java APIs for XML Messaging 1.3	JSR 67	⌂	
Java API for XML Registries (JAXR) 1.0	JSR 93	⌂	
Web application technologies			
Java Servlet 3.0	JSR 315	⌂	⌂
JavaServer Faces 2.0	JSR 314	⌂	⌂
JavaServer Pages 2.2/Expression Language 2.2	JSR 245	⌂	⌂
Standard Tag Library for JavaServer Pages (JSTL) 1.2	JSR 52	⌂	⌂
Debugging Support for Other Languages 1.0	JSR 45	⌂	⌂
Enterprise application technologies			
Contexts and Dependency Injection for Java (Web Beans 1.0)	JSR 299	⌂	
Dependency Injection for Java 1.0	JSR 330	⌂	
Bean Validation 1.0	JSR 303	⌂	⌂
Enterprise JavaBeans 3.1 (includes Interceptors 1.1)	JSR 318	⌂	
Java EE Connector Architecture 1.6	JSR 322	⌂	
Java Persistence 2.0	JSR 317	⌂	⌂

Table 22. *Java EE 6 support by profile (continued).*

A list of Java EE technologies, subdivided into sections for web services, web applications, enterprise applications, management and security, and Java EE-related specifications in Java SE. For each technology there is a specification reference, and an indication of whether the technology is supported by the full profile, and by the Liberty profile.

Technology	Specification reference	Full profile	Liberty profile
Common Annotations for the Java Platform 1.1	JSR 250	⌂	⌂(The Liberty profile does not support @ManagedBean.)
Java Message Service API 1.1	JSR 914	⌂	
Java Transaction API (JTA) 1.1	JSR 907	⌂	⌂
JavaMail 1.4	JSR 919	⌂	
Management and security technologies			
Java Authentication Service Provider Interface for Containers	JSR 196	⌂	
Java Authorization Contract for Containers 1.3	JSR 115	⌂	
Java EE Application Deployment 1.2	JSR 88	⌂	
J2EE Management 1.1	JSR 77	⌂	
Java EE-related specifications in Java SE			
Java API for XML Processing (JAXP) 1.3	JSR 206	⌂	⌂
Java Database Connectivity 4.0	JSR 221	⌂	⌂
Java Management Extensions (JMX) 2.0	JSR 255	⌂	⌂
JavaBeans Activation Framework (JAF) 1.1	JSR 925	⌂	⌂
Streaming API for XML (StAX) 1.0	JSR 173	⌂	⌂

Enterprise OSGi technologies

Table 23. Enterprise OSGi support by profile.

A list of enterprise OSGi technologies, subdivided into sections for blueprint, web, and other enterprise technologies. For each technology there is a specification reference, and an indication of whether the technology is supported by the full profile, and by the Liberty profile.

Technology	**Specification reference**	**full profile**	**Liberty profile**
Blueprint-related technologies			
Blueprint Container	R4.2 Enterprise Chapter 121	⌂	⌂
Blueprint Transactions		⌂	⌂
Blueprint Managed JPA		⌂	⌂
Blueprint Security		⌂	
Blueprint Resource References		⌂	
Web-related technologies			
Web Application Bundles	R4.2 Enterprise Chapter 128	⌂	⌂
JNDI	R4.2 Enterprise Chapter 126	⌂	⌂
JSP		⌂	⌂
JSTL		⌂	⌂
JSF		⌂	⌂
JAX-RS		⌂	⌂
Other enterprise technologies			
EJB Bundles		⌂(EJB levels earlier than 3.0 are not supported.)	
Remote Services	R4.2 Compendium Chapter 13	⌂	
SCA Configuration Type Specification	R4.2 Enterprise Chapter 129	⌂	
Remote Bundle Repositories		⌂	
SIP		⌂(SIP annotations are not supported.)	

Liberty profile externals support

External functions and resources of the Liberty profile can be used directly, and can be relied on to be in the next release. Internal or incidental aspects of the profile might change when you apply service, or upgrade to a future release.

What can I use directly in the profile and rely on being in the next release?

The following resources can be used directly and will continue to be available in the next release:

1. The documented application programming interfaces (APIs).
   - Write your code in accordance with the Liberty API documentation.[2].
   - Compile your code against the JAR files in the `wlp/dev` directories.
   - At run time, the application class loader has visibility to the API that is documented for the features in your server configuration. See the Liberty feature API documentation.[2]
2. The server configuration, including features.
3. Commands and scripts in the `wlp/bin` and `wlp/bin/tools` directories.
4. Client utilities in the `wlp/clients` directory.

What should I avoid dependencies on?

Do not build dependencies on incidental aspects of the product, or you might be impacted when applying service or upgrading to future releases. Examples of product internals that you should avoid relying on include, but are not restricted to, the following:

- The names of product binary jars, for example those in the `wlp/dev` directory. Compile your code against these JAR files by using the tools or using the `javac -extlibs` option.
- Direct use of the product binaries in the `/lib` directory. The only JAR files that can be directly invoked are in the `wlp/bin/tools` directory.
- Message text that is output by the server at run time.
- The layout of the product install, other than the `/wlp/bin` and `/wlp/dev` directories.
- Example and template files in the `wlp/templates` directory. These files might be modified when you apply service to your installation.
- Private or third party Java packages that are not explicitly exposed as APIs. These are not visible to the application class loader at run time.

What might be modified by applying service or an upgrade?

The contents of the following directories and their subdirectories might be modified when service or upgrade is applied. Do not make your own modifications to files in these locations, or they might be overwritten by product maintenance or upgrade:

- `wlp/bin`
- `wlp/clients`
- `wlp/dev`
- `wlp/lib`
- `wlp/templates`

No modifications will be made to the contents of the following directories. These are your files, and applying service or upgrade will not modify them:

- `wlp/etc` (where you might have added a `server.env` or `jvm.options` file).

[2] The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

- `wlp/usr` (the default location for user configuration and applications).
- Any non-default directory that you designate through the `WLP_USER_DIR` environment variable.

IBM i An exception for no modifications to `wlp/etc` is the file `wlp/etc/default.env`. This file is created when you install the Liberty profile on IBM iSeries® Platforms using the Installation Manager. The file is created or replaced by the `iAdmin POSTINSTALL` command during archive and Job Manager installs. The `iAdmin` command is located in the `wlp/lib/native/os400/bin` directory. See Liberty profile: iAdmin command.

Liberty profile: Server configuration

The Liberty profile is configured by exception. The runtime environment operates from a set of built-in configuration defaults, and you only specify the overrides that you need to those defaults. You do this by editing an XML configuration file; either `server.xml` or another XML file that is included in `server.xml` at run time.

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server. They are the primary mechanism that makes the server composable. The list of features that you specify in the server configuration provides a functional server.

When you first install and start the server, a feature manager and a default server configuration are available to you:

- By default, a server contains the `jsp-2.2` feature, to support servlet and JSP applications. You use the feature manager to add the features that you need.
- Server configuration is by exception. When you specify the features that you need, the default configuration of those features provides a rich environment that is designed to cover most common requirements, so you only need to specify changes from the default.

The configuration has the following characteristics:

- Described in XML files.
- Human-readable, and editable in a text editor.
- Small, easy to back up, and easy to copy to another system.
- Shareable across an application development team.
- Composable, so that features can easily add their own configuration to the system.
- Extensibly-typed, so you don't have to modify the current configuration to work with later versions of the runtime environment.
- Dynamically responsive to updates.
- Forgiving, so that missing values are assumed and unrecognized properties are ignored.

You can also use a `bootstrap.properties` file to specify properties that need to be available before the main configuration is processed, and to define variables that are used in the main configuration.

Liberty profile: Configuration elements in the server.xml file

The application server configuration is described in a series of elements in the `server.xml` configuration file. Each element has one or more attributes or sub-elements. This topic contains details of the possible elements, attributes and sub-elements that can be configured.

List of elements in the server.xml configuration file.

- “activedLdapFilterProperties” on page 197
- “administrator-role” on page 198
- “application” on page 198
- “application-bnd” on page 199

- “applicationMonitor” on page 200
- “authCache” on page 201
- “authData” on page 202
- “authentication” on page 202
- “basicRegistry” on page 202
- “bundleRepository” on page 203
- “channelfw” on page 203
- “classloader” on page 204
- “config” on page 205
- “connectionManager” on page 206
- “customLdapFilterProperties” on page 207
- “dataSource” on page 208
- “domino50LdapFilterProperties” on page 211
- “edirectoryLdapFilterProperties” on page 211
- “executor” on page 212
- “featureManager” on page 213
- “fileset” on page 214
- “httpClassification” on page 214
- “httpEncoding” on page 215
- “httpEndpoint” on page 223
- “httpOptions” on page 225
- “httpSession” on page 225
- “httpSessionDatabase” on page 229
- “idsLdapFilterProperties” on page 233
- “iplanetLdapFilterProperties” on page 233
- “jaasLoginContextEntry” on page 234
- “jaasLoginModule” on page 234
- “jdbcDriver” on page 235
- “jndiEntry” on page 236
- “jpa” on page 236
- “jspEngine” on page 237
- “keyStore” on page 238
- “ldapRegistry” on page 238
- “library” on page 242
- “logging” on page 243
- “ltpa” on page 245
- “mimeTypes” on page 245
- “monitor” on page 245
- “nativeTransactionManager” on page 245
- “netscapeLdapFilterProperties” on page 246
- “pluginConfiguration” on page 246
- “properties” on page 247
- “properties.datadirect.sqlserver” on page 248
- “properties.db2.i.native” on page 255
- “properties.db2.i.toolbox” on page 261

- “properties.db2.jcc” on page 271
- “properties.derby.client” on page 278
- “properties.derby.embedded” on page 280
- “properties.informix” on page 281
- “properties.informix.jcc” on page 289
- “properties.microsoft.sqlserver” on page 295
- “properties.oracle” on page 298
- “properties.sybase” on page 299
- “quickStartSecurity” on page 301
- “safAuthorization” on page 301
- “safCredentials” on page 301
- “safRegistry” on page 302
- “safRoleMapper” on page 302
- “securewayLdapFilterProperties” on page 302
- “ssl” on page 303
- “sslDefault” on page 303
- “sslOptions” on page 303
- “tcpOptions” on page 304
- “transaction” on page 304
- “trustAssociation” on page 306
- “virtualHost” on page 307
- “webAppSecurity” on page 308
- “webContainer” on page 310
- “wlmClassification” on page 316
- “zosWorkloadManager” on page 317

activedLdapFilterProperties

Specifies the list of default Active Directory LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.actived.

`userFilter`

An LDAP filter clause for searching the user registry for users.

string

(&(sAMAccountName=%v)(objectcategory=user))

true

`groupFilter`

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(objectcategory=group))

true

`userIdMap`

An LDAP filter that maps the name of a user to an LDAP entry.

string

user:sAMAccountName

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap

An LDAP filter that identifies user to group memberships.

string

memberof:member

true

administrator-role

A collection of users and/or groups assigned the server administrator role. PID is com.ibm.ws.management.security.role.administrator.

user

User assigned a role.

string

false

group

Group assigned a role.

string

false

application

Defines the properties of an application. PID is com.ibm.ws.app.manager.

location

Location of an application expressed as an absolute path or a path relative to the server-level apps directory.

string

true

name

Name of an application.

string

false

type

Type of application archive.

string

false

context-root

Context root of an application.

string

false

autoStart

Indicates whether or not the server should start the application automatically when the server starts.

boolean

true

false

application-bnd

Binds general deployment information included in the application to specific resources. PID is com.ibm.ws.javaee.dd.appbnd, and it is the child of complex type "application".

version

Version of the application bindings specification.

string

false

security-role

A role that is mapped to users and groups in a domain user registry.

name

Name of a security role.

string

true

user

A user possessing a security role.

name

Name of a user possessing a security role.

string

true

access-id

A user access ID in the general form user:realmName/userUniqueId. A value will be generated if one is not specified.

string

false

false

group

A group possessing a security role.

name

Name of a group possessing a security role.

string

true

access-id

Group access ID

string

false

false

special-subject

Name of a special-subject possessing a security role.

type

One of the following special subject types: ALL_AUTHENTICATED_USERS, EVERYONE.

string

EVERYONE

ALL_AUTHENTICATED_USERS

All authenticated users

true

false

run-as

ID and password of a user required to access a bean from another bean.

userid

ID of a user required to access a bean from another bean.

string

true

password

Password of a user required to access a bean from another bean. The value can be stored in clear text or encoded form. To encode the password, use the securityUtility tool with the encode option.

password (string)

false

false

false

applicationMonitor

Defines how the server responds to application additions, updates, and deletions. PID is com.ibm.ws.app.manager.monitor.

pollingRate

Rate at which the server checks for application additions, updates, and deletions. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

500ms

false

dropins

Location of the application drop-in directory expressed as an absolute path or a path relative to the server directory.

string

dropins

false

`dropinsEnabled`
Monitor the drop-in directory for application additions, updates, and deletions.

boolean

true

false

`updateTrigger`
Application update method or trigger.

string

polled

polled Server will scan for application changes at the polling interval and update any applications that have detectable changes.

mbean
Server will only update applications when prompted by an MBean called by an external program such as an integrated development environment or a management application.

disabled
Disables all update monitoring. Application changes will not be applied while the server is running.

false

authCache

Controls the operation of the authentication cache service. PID is com.ibm.ws.security.authentication.cache.

`initialSize`
Initial number of entries supported by the authentication cache.

int

50

false

`maxSize`
Maximum number of entries supported by the authentication cache.

int

25000

false

`timeout`
Amount of time after which an entry in the cache will be removed. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

600s

false

`allowBasicAuthLookup`
Allow lookup by user ID and hashed password.

boolean

true

false

authData

Authentication data for a JCA connection to an Enterprise Information System (EIS). PID is com.ibm.ws.security.jca.internal.authdata.config.

`user`

Name of the user to use when connecting to the EIS.

string

true

`password`

Password of the user to use when connecting to the EIS. The value can be stored in clear text or encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool with the encode option.

password (string)

true

authentication

Controls the built-in authentication service configuration. PID is com.ibm.ws.security.authentication.

`cacheEnabled`

Enables the authentication cache.

boolean

true

true

`allowHashtableLoginWithIdOnly`

Allow an application to login with just an identity in the hashtable properties. Use this option only when you have applications that require this and have other means to validate the identity.

boolean

false

false

basicRegistry

A simple XML-based user registry. PID is com.ibm.ws.security.registry.basic.config.

`realm`

The realm name represents the user registry.

string

BasicRegistry

true

`user`

A user in a Basic User Registry.

`name`

Name of a user in a Basic User Registry.

string

true

password
Password of a user in a Basic User Registry. The value can be stored in clear text or encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool with the encode option.

password (string)

true

false

group
A group in a Basic User Registry.

name
Name of a group in a Basic User Registry.

string

true

member
A member of a Basic User Registry group.

name
Name of a user in a Basic User Registry group.

string

true

false

false

bundleRepository

EBA bundle repository service. PID is com.ibm.ws.eba.bundle.repository.

filesetRef
Space separated list of fileset references

List of configuration IDs of type fileset (comma-separated string).

false

fileset
Space separated list of fileset references

Element of type fileset.

false

channelfw

Defines channel and chain management settings. PID is com.ibm.ws.channelfw.

chainStartRetryInterval
Time interval between start retries. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

5s

false

`chainStartRetryAttempts`

Number of retry attempts to make per chain.

int

60

false

`chainQuiesceTimeout`

Default amount of time to wait while quiescing chains. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

30s

false

`warningWaitTime`

Amount of time to wait before notifying of a missing factory configuration. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

10s

false

classloader

Classloader Service. PID is com.ibm.ws.classloading.classloader, and it is the child of complex type “application”.

`delegation`

Controls whether parent classloader is used before or after this classloader.

string

parentFirst

parentFirst

Parent first

parentLast

Parent last

false

`privateLibraryRef`

List of library references. Library class instances are unique to this classloader, independent of class instances from other classloaders.

List of configuration IDs of type library (comma-separated string).

false

`commonLibraryRef`

List of library references. Library class instances are shared with other classloaders.

List of configuration IDs of type library (comma-separated string).

false

apiTypeVisibility

The types of API package this class loader will be able to see, as a comma-separated list of any combination of the following: spec, ibm-api, ibm-spi, third-party.

string

spec,ibm-api

false

privateLibrary

List of library references. Library class instances are unique to this classloader, independent of class instances from other classloaders.

Element of type library.

false

commonLibrary

List of library references. Library class instances are shared with other classloaders.

Element of type library.

false

config

Defines how the server processes configuration information. PID is com.ibm.ws.config.

onError

Action to take after a incurring a configuration error.

onError

string

WARN

WARN
Server will issue warning and error messages when it incurs a configuration error.

FAIL Server will issue a warning or error message on the first error occurrence and then stop the server.

IGNORE
Server will not issue any warning and error messages when it incurs a configuration error.

true

monitorInterval

Rate at which the server checks for configuration updates. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

500ms

false

updateTrigger

Configuration update method or trigger.

string

polled

polled Server will scan for changes at the polling interval on all the configuration files and update the runtime configuration with the changes detected.

mbean
Server will only update the configuration when prompted by an MBean called by an external program such as an integrated development environment or a management application.

disabled
Disables all update monitoring. Configuration changes will not be applied while the server is running.

false

connectionManager

Connection Manager configuration. PID is com.ibm.ws.jca.connectionManager.

`agedTimeout`
Amount of time before a physical connection can be discarded by pool maintenance. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

-1

false

`connectionTimeout`
Amount of time after which a connection request times out. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

30s

false

`maxIdleTime`
Amount of time after which an unused or idle connection can be discarded during pool maintenance, if doing so does not reduce the pool below the minimum size. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

30m

false

`maxPoolSize`
Maximum number of physical connections for a pool. A value of 0 means unlimited.

int

50

false

`minPoolSize`
Minimum number of physical connections to maintain in the pool. The pool is not pre-populated. Aged timeout can override the minimum.

int

false

purgePolicy
Specifies which connections to destroy when a stale connection is detected in a pool.

string

EntirePool

EntirePool
When a stale connection is detected, all connections in the pool are marked stale, and when no longer in use, are closed.

FailingConnectionOnly
When a stale connection is detected, only the connection which was found to be bad is closed.

ValidateAllConnections
When a stale connection is detected, connections are tested and those found to be bad are closed.

false

reapTime
Amount of time between runs of the pool maintenance thread. A value of -1 disables pool maintenance. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

3m

false

maxConnectionsPerThread
Limits the number of open connections on each thread.

int

false

numConnectionsPerThreadLocal
Caches the specified number of connections for each thread.

int

false

customLdapFilterProperties

Specifies the list of default Custom LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.custom.

userFilter
An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=ePerson))

true

groupFilter
An LDAP filter clause for search the user registry for groups.

string

(&(cn=
%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))

true

userIdMap

An LDAP filter that maps the name of a user to an LDAP entry.

string

*:uid

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap

An LDAP filter that identifies user to group memberships.

string

ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember

true

dataSource

Defines a data source configuration. PID is com.ibm.ws.jdbc.dataSource.

jndiName

JNDI name for a data source.

string

true

jdbcDriverRef

JDBC driver for a data source.

Configuration ID of type jdbcDriver (string).

false

connectionManagerRef

Connection manager for a data source.

Configuration ID of type connectionManager (string).

false

type

Type of data source.

string

javax.sql.XADataSource

javax.sql.ConnectionPoolDataSource

javax.sql.DataSource

false

connectionSharing

Specifies how connections are matched for sharing.

string

MatchOriginalRequest

MatchOriginalRequest
When sharing connections, match based on the original connection request.

MatchCurrentState
When sharing connections, match based on the current state of the connection.

false

`isolationLevel`
Default transaction isolation level.

string

TRANSACTION_READ_UNCOMMITTED
Dirty reads, non-repeatable reads and phantom reads can occur.

TRANSACTION_READ_COMMITTED
Dirty reads are prevented; non-repeatable reads and phantom reads can occur.

TRANSACTION_REPEATABLE_READ
Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

TRANSACTION_SERIALIZABLE
Dirty reads, non-repeatable reads and phantom reads are prevented.

TRANSACTION_SNAPSHOT
Snapshot isolation for Microsoft SQL Server JDBC Driver and DataDirect Connect for JDBC driver.

false

`statementCacheSize`
Maximum number of cached statements per connection.

int

10

false

`transactional`
Enable participation in transactions that are managed by the application server.

boolean

true

false

`beginTranForResultSetScrollingAPIs`
Attempt transaction enlistment when result set scrolling interfaces are used.

boolean

true

false

`beginTranForVendorAPIs`
Attempt transaction enlistment when vendor interfaces are used.

boolean

true

false

commitOrRollbackOnCleanup

Determines how to clean up connections that might be in a database unit of work (AutoCommit=false) when the connection is closed or returned to the pool.

string

commit

Clean up the connection by committing.

rollback

Clean up the connection by rolling back.

false

queryTimeout

Default query timeout for SQL statements. In a JTA transaction, syncQueryTimeoutWithTransactionTimeout can override this default. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

recoveryAuthDataRef

Authentication data for transaction recovery.

Configuration ID of type authData (string).

false

syncQueryTimeoutWithTransactionTimeout

Use the time remaining (if any) in a JTA transaction as the default query timeout for SQL statements.

boolean

false

false

supplementalJDBCTrace

Supplements the JDBC driver trace that is logged when JDBC driver trace is enabled in bootstrap.properties. JDBC driver trace specifications include: com.ibm.ws.database.logwriter, com.ibm.ws.db2.logwriter, com.ibm.ws.derby.logwriter, com.ibm.ws.informix.logwriter, com.ibm.ws.oracle.logwriter, com.ibm.ws.sqlserver.logwriter, com.ibm.ws.sybase.logwriter.

boolean

false

jdbcDriver

JDBC driver for a data source.

Element of type jdbcDriver.

false

connectionManager

Connection manager for a data source.

Element of type connectionManager.

false

recoveryAuthData

Authentication data for transaction recovery.

Element of type authData.

false

domino50LdapFilterProperties

Specifies the list of default Domino® LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.domino50.

userFilter

An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=Person))

true

groupFilter

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(objectclass=dominoGroup))

true

userIdMap

An LDAP filter that maps the name of a user to an LDAP entry.

string

person:uid

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap

An LDAP filter that identifies user to group memberships.

string

dominoGroup:member

true

edirectoryLdapFilterProperties

Specifies the list of EDirectory LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.eDirectory.

userFilter

An LDAP filter clause for searching the user registry for users.

string

(&(cn=%v)(objectclass=Person))

true

groupFilter

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(objectclass=groupOfNames))

true

userIdMap

An LDAP filter that maps the name of a user to an LDAP entry.

string

person:cn

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap

An LDAP filter that identifies user to group memberships.

string

groupOfNames:member

true

executor

Defines threading and execution settings for the server. PID is com.ibm.ws.threading.

name

Name of the executor for which the thread is performing work.

string

Default Executor

false

maxThreads

Maximum number of threads that can be associated with the executor. If greater than 0, this value must be greater than or equal to the value of coreThreads. If the value of maxThreads is less than or equal to 0, the maximum number of threads is unbounded.

int

-1

false

coreThreads

Steady-state or core number of threads to associate with the executor. The number of threads associated with the executor will quickly grow to this number. If this value is less than 0, a default value is used. This default value is calculated based on the number of hardware threads on the system.

int

-1

false

keepAlive

Amount of time to keep an idle thread in the pool before allowing it to terminate. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

60s

false

`stealPolicy`

The work-stealing policy to employ. The options for this policy determine how work is queued, and how threads obtain queued work.

string

STRICT

STRICT
All threads that generate work own a local work pile. Threads that are associated with the executor take work from other threads when the local work pile is exhausted.

LOCAL
A global work queue is used for work that is generated by threads that are not associated with the executor. Work generated by threads associated with the executor are placed on a local work pile. This work pile is owned by the generating thread, unless another thread steals it. Threads that are associated with the executor take work associated with other threads if the local work pile is empty and there is no work on the global work queue.

NEVER
A global work queue is used to feed work to threads that are associated with the executor. No stealing will occur.

false

`rejectedWorkPolicy`

Policy to employ when the executor is unable to stage work for execution.

string

ABORT

ABORT
Raise an exception.

CALLER_RUNS
Execute the work immediately on the caller's thread.

false

featureManager

Defines how the server loads features. PID is com.ibm.ws.kernel.feature.

`onError`

Action to take after a failure to load a feature.

onError

string

WARN

WARN
Server will issue warning and error messages when it incurs a feature configuration error.

FAIL Server will issue a warning or error message on the first feature configuration error occurrence and then stop the server.

IGNORE
Server will not issue any warning and error messages when it incurs a feature configuration error.

true

`feature`
string

false

fileset

Fileset Service. PID is com.ibm.ws.kernel.metatype.helper.fileset.

`dir`
The base directory to search for files.

string

.

true

`caseSensitive`
Boolean to indicate whether or not the search should be case sensitive (default: true).

boolean

true

false

`includes`
The comma or space separated list of file name patterns to include in the search results (default: *).

string

*

false

`excludes`
The comma or space separated list of file name patterns to exclude from the search results, by default no files are excluded.

string

false

`scanInterval`
Scanning interval to check the fileset for changes as a long with a time unit suffix h-hour, m-minute, s-second, ms-millisecond (e.g. 2ms or 5s). Disabled (scanInterval=0) by default. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

0

false

httpClassification

zos.wlm.httpclassification.config.description. PID is com.ibm.ws.zos.wlm.httpclassification, and it is the child of complex type “wlmClassification”.

`transactionClass`
Defines the priority

string

false

host
Defines which host to map the transaction class to

string

*

false

port
Defines which port to map the transaction class to

string

*

false

resource
Defines the URI to use when mapping transaction class

string

*

false

method
Defines the HTTP method to map to

string

*

false

httpEncoding

Configuration properties for the HTTP Transport Encoding service. PID is com.ibm.ws.transport.http.encoding.

converter.Shift_JIS
Converter for Shift_JIS

string

Cp943C

false

converter.EUC-JP
Converter for EUC-JP

string

Cp33722C

false

converter.EUC-KR
Converter for EUC-KR

string

Cp970

false

converter.EUC_KR
Converter for EUC_KR

string

Cp970

false

converter.EUC-TW
Converter for EUC-TW

string

Cp964

false

converter.Big5
Converter for Big5

string

Cp950

false

converter.GB2312
Converter for GB2312

string

EUC_CN

false

converter.ISO-2022-KR
Converter for ISO-2022-KR

string

ISO2022KR

false

encoding.en
Encoding for 'en' locale

string

ISO-8859-1

false

encoding.fr
Encoding for 'fr' locale

string

ISO-8859-1

false

encoding.de
Encoding for 'de' locale

string

ISO-8859-1

false

encoding.es

Encoding for 'es' locale

string

ISO-8859-1

false

encoding.pt

Encoding for 'pt' locale

string

ISO-8859-1

false

encoding.da

Encoding for 'da' locale

string

ISO-8859-1

false

encoding.ca

Encoding for 'ca' locale

string

ISO-8859-1

false

encoding.fi

Encoding for 'fi' locale

string

ISO-8859-1

false

encoding.it

Encoding for 'it' locale

string

ISO-8859-1

false

encoding.nl

Encoding for 'nl' locale

string

ISO-8859-1

false

encoding.no

Encoding for 'no' locale

string

ISO-8859-1

false

`encoding.sv`

Encoding for 'sv' locale

string

ISO-8859-1

false

`encoding.is`

Encoding for 'is' locale

string

ISO-8859-1

false

`encoding.eu`

Encoding for 'eu' locale

string

ISO-8859-1

false

`encoding.cs`

Encoding for 'cs' locale

string

ISO-8859-2

false

`encoding.hr`

Encoding for 'hr' locale

string

ISO-8859-2

false

`encoding.hu`

Encoding for 'hu' locale

string

ISO-8859-2

false

`encoding.lt`

Encoding for 'lt' locale

string

ISO-8859-2

false

`encoding.pl`

Encoding for 'pl' locale

string

ISO-8859-2

false

encoding.sh
Encoding for 'sh' locale

string

ISO-8859-2

false

encoding.sk
Encoding for 'sk' locale

string

ISO-8859-2

false

encoding.sl
Encoding for 'sl' locale

string

ISO-8859-2

false

encoding.sq
Encoding for 'sq' locale

string

ISO-8859-2

false

encoding.fo
Encoding for 'fo' locale

string

ISO-8859-2

false

encoding.ro
Encoding for 'ro' locale

string

ISO-8859-2

false

encoding.mt
Encoding for 'mt' locale

string

ISO-8859-3

false

encoding.et
Encoding for 'et' locale

string

ISO-8859-4

false

encoding.lv
Encoding for 'lv' locale

string

ISO-8859-4

false

encoding.be
Encoding for 'be' locale

string

ISO-8859-5

false

encoding.bg
Encoding for 'bg' locale

string

ISO-8859-5

false

encoding.mk
Encoding for 'mk' locale

string

ISO-8859-5

false

encoding.ru
Encoding for 'ru' locale

string

ISO-8859-5

false

encoding.sr
Encoding for 'sr' locale

string

ISO-8859-5

false

encoding.uk
Encoding for 'uk' locale

string

ISO-8859-5

false

encoding.ar
Encoding for 'ar' locale

string

ISO-8859-6

false

`encoding.fa`

Encoding for 'fa' locale

string

ISO-8859-6

false

`encoding.ms`

Encoding for 'ms' locale

string

ISO-8859-6

false

`encoding.el`

Encoding for 'el' locale

string

ISO-8859-7

false

`encoding.iw`

Encoding for 'iw' locale

string

ISO-8859-8

false

`encoding.he`

Encoding for 'he' locale

string

ISO-8859-8

false

`encoding.ji`

Encoding for 'ji' locale

string

ISO-8859-8

false

`encoding.yi`

Encoding for 'yi' locale

string

ISO-8859-8

false

`encoding.tr`

Encoding for 'tr' locale

string

ISO-8859-9

false

`encoding.th`

Encoding for 'th' locale

string

windows-874

false

`encoding.vi`

Encoding for 'vi' locale

string

windows-1258

false

`encoding.ja`

Encoding for 'ja' locale

string

Shift_JIS

false

`encoding.ko`

Encoding for 'ko' locale

string

EUC-KR

false

`encoding.zh`

Encoding for 'zh' locale

string

GB2312

false

`encoding.zh_TW`

Encoding for 'zh_TW' locale

string

Big5

false

`encoding.hy`

Encoding for 'hy' locale

string

UTF-8

false

`encoding.ka`

Encoding for 'ka' locale

string

UTF-8

false

encoding.hi
Encoding for 'hi' locale

string

UTF-8

false

encoding.mr
Encoding for 'mr' locale

string

UTF-8

false

encoding.sa
Encoding for 'sa' locale

string

UTF-8

false

encoding.ta
Encoding for 'ta' locale

string

UTF-8

false

encoding.bn
Encoding for 'bn' locale

string

UTF-8

false

httpEndpoint

Configuration properties for an HTTP endpoint. PID is com.ibm.ws.http.

httpOptionsRef
HTTP protocol options for the endpoint.

defaultHTTPVar

Configuration ID of type httpOptions (string).

false

sslOptionsRef
SSL protocol options for the endpoint.

defaultSSLVar

Configuration ID of type sslOptions (string).

false

tcpOptionsRef
TCP protocol options for the endpoint.

Configuration ID of type tcpOptions (string).

defaultTCPOptions

false

enabled

Toggle the availability of an endpoint. When true, this endpoint will be activated by the dispatcher to handle HTTP requests.

boolean

true

false

host

IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource. Use '*' for all available network interfaces.

string

localhost

false

httpPort

The port used for client HTTP requests. Use -1 to disable this port.

int

false

httpsPort

The port used for client HTTP requests secured with SSL (https). Use -1 to disable this port.

int

false

virtualHost

ID of a configured virtual host. All endpoints are associated with the 'default_host' virtual host by default.

string

default_host

false

httpOptions

HTTP protocol options for the endpoint.

defaultHTTPVar

Element of type httpOptions.

false

sslOptions

SSL protocol options for the endpoint.

defaultSSLVar

Element of type sslOptions.

false

tcpOptions

TCP protocol options for the endpoint.

Element of type tcpOptions.

defaultTCPOptions

false

httpOptions

HTTP protocol configuration. PID is com.ibm.ws.http.options.

keepAliveEnabled

Enables persistent connections (HTTP keepalive). If true, connections are kept alive for reuse by multiple sequential requests and responses. If false, connections are closed after the response is sent.

boolean

true

false

maxKeepAliveRequests

Maximum number of persistent requests that are allowed on a single HTTP connection if persistent connections are enabled. A value of -1 means unlimited.

int

100

false

persistTimeout

Amount of time that a socket will be allowed to remain idle between requests. This setting only applies if persistent connections are enabled. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

30s

false

readTimeout

Amount of time to wait for a read request to complete on a socket after the first read occurs. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

60s

false

writeTimeout

Amount of time to wait on a socket for each portion of the response data to be transmitted. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

60s

false

httpSession

Configuration for HTTP session management. PID is com.ibm.ws.session.

storageRef
The ID of the persistent storage location for session data. If no storage location is defined, session data will be stored in the local application server's memory.

string

false

sslTrackingEnabled
Specifies that session tracking uses Secure Sockets Layer (SSL) information as a session ID.

boolean

false

false

urlRewritingEnabled
Specifies that the session management facility uses rewritten URLs to carry the session IDs.

boolean

false

false

protocolSwitchRewritingEnabled
Adds the session ID to a URL when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP.

boolean

false

false

cookiesEnabled
Specifies that session tracking uses cookies to carry session IDs.

boolean

true

false

cookieName
A unique name for a session management cookie.

string

JSESSIONID

false

cookieDomain
Domain field of a session tracking cookie.

string

false

cookieMaxAge
Maximum amount of time that a cookie can reside on the client browser.

int

-1

false

cookiePath

A cookie is sent to the URL designated in the path.

string

/

false

cookieSecure

Specifies that the session cookies include the secure field.

boolean

false

false

cookieHttpOnly

Specifies that session cookies include the HttpOnly field. Browsers that support the HttpOnly field do not enable cookies to be accessed by client-side scripts. Using the HttpOnly field will help prevent cross-site scripting attacks.

boolean

true

false

maxInMemorySessionCount

Maximum number of sessions to maintain in memory for each web module.

int

1000

false

allowOverflow

Allows the number of sessions in memory to exceed the value of the Max in-memory session count property.

boolean

true

false

invalidationTimeout

Amount of time a session can go unused before it is no longer valid.

long

1800

false

securityIntegrationEnabled

Enables security integration, which causes the session management facility to associate the identity of users with their HTTP sessions.

boolean

true

false

idLength

Length of the session identifier.

int

23

false

`useContextRootAsCookiePath`
Specifies that the cookie path equals the context root of the web module instead of /

boolean

false

false

`alwaysEncodeUrl`
The Servlet 2.5 specification specifies to not encode the URL on a response.encodeURL call if it is not necessary. To support backward compatibility when URL encoding is enabled, set this property to true to call the encodeURL method. The URL is always encoded, even if the browser supports cookies.

boolean

false

false

`cloneId`
The clone ID of the cluster member. Within a cluster, this ID must be unique to maintain session affinity. When set, this name overwrites the default name generated by the server.

string

false

`cloneSeparator`
The single character used to separate the session ID from the clone ID in session cookies. The default value should usually be used. On some Wireless Application Protocol (WAP) devices, a colon (:) is not allowed, so a plus sign (+) should be used instead. Different values should rarely be used. You should understand the clone character requirements of other products running on your system before using this property to change the clone separator character. The fact that any character can be specified as the value for this property does not imply that the character you specify will function correctly. This fact also does not imply that IBM is responsible for fixing any problem that might arise from using an alternative character.

string

:

false

`debugCrossover`
Enable this option to perform additional checks to verify that only the session associated with the request is accessed or referenced, and log messages if any discrepancies are detected. Disable this option to skip the additional checks.

boolean

false

false

`forceInvalidationMultiple`
If your requests normally are not bound by a response time limit, specify 0 to indicate that the session manager should wait indefinitely until a request is complete before attempting to invalidate the session. Otherwise, set this property to a positive integer to delay the invalidation of active sessions. Active timed out sessions will not be invalidated by the first invalidation interval pass, but will be invalidated

by the interval pass based on this value. For example, a value of 2 would invalidate an active session on the second invalidation interval pass after the session timeout has expired.

int

3

false

idReuse

In a multi-JVM environment that is not configured for session persistence, setting this property to "true" enables the session manager to use the same session information for all of a user's requests even if the web applications that are handling these requests are governed by different JVMs. The default value for this property is false. Set this property to true if you want to enable the session manager to use the session ID sent from a browser to preserve session data across web applications that are running in an environment that is not configured for session persistence.

boolean

false

false

noAdditionalInfo

Forces removal of information that is not needed in session identifiers.

boolean

false

false

reaperPollInterval

The wake-up interval, in seconds, for the process that removes invalid sessions. The minimum value is 30 seconds. If a value less than the minimum is entered, an appropriate value is automatically determined and used. This value overrides the default installation value, which is between 30 and 360 seconds, based off the session timeout value. Because the default session timeout is 30 minutes, the reaper interval is usually between 2 and 3 minutes.

long

-1

false

rewriteId

Use this property to change the key used with URL rewriting.

string

jsessionid

false

securityUserIgnoreCase

Indicates that the session security identity and the client security identity should be considered a match even if their cases are different. For example, when this property is set to true, the session security identity USER1 matches the client security identities User1 and user1.

boolean

false

false

httpSessionDatabase

Controls how HTTP sessions are persisted to a database. PID is com.ibm.ws.session.db.

`dataSourceRef`
The ID of the data source the session manager should use to persist HTTP session data.

string

true

`useMultiRowSchema`
When enabled, each session data attribute is placed in a separate row in the database, allowing larger amounts of data to be stored for each session. This configuration can yield better performance when session attributes are very large and few changes are required to the session attributes. When disabled, all session data attributes are placed in the same row for each session.

boolean

false

false

`db2RowSize`
Table space page size configured for the sessions table, if using a DB2 database. Increasing this value can improve database performance in some environments.

string

4KB

4KB Use the default table space page size of 4 KB. You do not need to create a DB2 buffer pool or table space, and you do not need to specify a table space name.

8KB Use a table space page size of 8 KB. You must additionally create a DB2 buffer pool and table space, and specify 8KB as the page size for both. You must also specify the name of the table space you created.

16KB Use a table space page size of 16 KB. You must additionally create a DB2 buffer pool and table space, and specify 16KB as the page size for both. You must also specify the name of the table space you created.

32KB Use a table space page size of 32 KB. You must additionally create a DB2 buffer pool and table space, and specify 32KB as the page size for both. You must also specify the name of the table space you created.

false

`tableSpaceName`
Table space to be used for the sessions table. This value is only required when the DB2 Row Size is greater than 4KB.

string

false

`scheduleInvalidation`
Enable this option to reduce the number of database updates required to keep the HTTP sessions alive. Specify the two hours of a day when there is the least activity in the application server. When this option is disabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions.

boolean

false

false

`scheduleInvalidationFirstHour`

Indicates the first hour during which the invalidated sessions are cleared from the persistent store. Specify this value as an integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

int

0

false

`scheduleInvalidationSecondHour`

Indicates the second hour during which the invalidated sessions are cleared from the persistent store. Specify this value as an integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

int

0

false

`writeFrequency`

Specifies when session data is written to the persistent store. By default, session data is written to the persistent store after the servlet completes execution. Changing this value can improve performance in some environments.

string

END_OF_SERVLET_SERVICE

END_OF_SERVLET_SERVICE
Session data is written to the persistent store after the servlet completes execution.

MANUAL_UPDATE
A programmatic sync on the IBMSession object is required to write the session data to the persistent store.

TIME_BASED_WRITE
Session data is written to the persistent store based on the specified write interval value.

false

`writeInterval`

Number of seconds that should pass before writing session data to the persistent store. The default is 120 seconds. This value is only used when a time based write frequency is enabled.

int

120

false

`writeContents`

Specifies how much session data should be written to the persistent store. By default, only updated attributes are written, but all attributes can be written instead (regardless of whether or not they changed).

string

ONLY_UPDATED_ATTRIBUTES

ONLY_UPDATED_ATTRIBUTES
Only updated attributes are written to the persistent store.

ALL_SESSION_ATTRIBUTES
All attributes are written to the persistent store.

false

noAffinitySwitchBack
Set this property to "true" to maintain affinity to the new member even after original one comes back up. When a cluster member fails, its requests routed to a different cluster member, and sessions are activated in that other member. Thus, session affinity is maintained to the new member, and when failed cluster member comes back up, the requests for sessions that were created in the original cluster member are routed back to it. Allowed values are true or false, with the default being false. Set this property to true when you have distributed sessions configured with time-based write. Note that this property has no affect on the behavior when distributed sessions is not enabled.

boolean

false

false

onlyCheckInCacheDuringPreInvoke
A value of true indicates that the last access time of a session should only be updated if a request gets the session. A value of false indicates that the last access time of a session should be updated upon every request. Changing this value can improve performance in some environments.

boolean

false

false

optimizeCacheIdIncrements
If the user's browser session is moving back and forth across multiple web applications, you might see extra persistent store activity as the in-memory sessions for a web module are refreshed from the persistent store. As a result, the cache IDs are continually increasing and the in-memory session attributes are overwritten by those of the persistent copy. Set this property to true if you want to prevent the cache IDs from continually increasing. A value of true indicates that the session manager should assess whether the in-memory session for a web module is older than the copy in persistent store. If the configuration is a cluster, ensure that the system times of each cluster member are as identical as possible.

boolean

true

false

tableName
The database table name.

string

sessions

false

useInvalidatedId
Set this property to "true" to reuse the incoming ID if the session with that ID was recently invalidated. This is a performance optimization because it prevents checking the persistent store.

boolean

true

false

useOracleBlob
Set this property to "true" to create the database table using the Binary Large Object (BLOB) data type for the medium column. This value increases performance of persistent sessions when Oracle

databases are used. Due to an Oracle restriction, BLOB support requires use of the Oracle Call Interface (OCI) database driver for more than 4000 bytes of data. You must also ensure that a new sessions table is created before the server is restarted by dropping your old sessions table or by changing the datasource definition to reference a database that does not contain a sessions table.

boolean

false

false

idsLdapFilterProperties

Specifies the list of default IBM Directory Server LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.ids.

`userFilter`

An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=ePerson))

true

`groupFilter`

An LDAP filter clause for search the user registry for groups.

string

(&(cn=
%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))

true

`userIdMap`

An LDAP filter that maps the name of a user to an LDAP entry.

string

*:uid

true

`groupIdMap`

groupIdFilter.desc

string

*:cn

true

`groupMemberIdMap`

An LDAP filter that identifies user to group memberships.

string

ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember

true

iplanetLdapFilterProperties

Specifies the list of default IPlanet LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.iPlanet.

`userFilter`

An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=inetOrgPerson))

true

groupFilter
An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(objectclass=ldapsubentry))

true

userIdMap
An LDAP filter that maps the name of a user to an LDAP entry.

string

inetOrgPerson:uid

true

groupIdMap
groupIdFilter.desc

string

*:cn

true

groupMemberIdMap
An LDAP filter that identifies user to group memberships.

string

nsRole:nsRole

true

jaasLoginContextEntry

The JAAS login context entry configuration. PID is com.ibm.ws.security.authentication.internal.jaas.jaasLoginContextEntry.

name
Name of a JAAS configuration entry.

string

true

loginModuleRef
A reference to the ID of a JAAS login module.

string

true

jaasLoginModule

A login module in the JAAS configuration. PID is com.ibm.ws.security.authentication.internal.jaas.jaasLoginModuleConfig.

className
Fully-qualified package name of the JAAS login module class.

string

true

`controlFlag`

The login module's control flag. Valid values are REQUIRED, REQUISITE, SUFFICIENT, and OPTIONAL.

string

REQUIRED

REQUIRED

This LoginModule is REQUIRED as per the JAAS specification. The LoginModule is required to succeed.

REQUISITE

controlFlag.REQUISITE

SUFFICIENT

This LoginModule is SUFFICIENT as per the JAAS specification. The LoginModule is not required to succeed. If authentication is successful, no other LoginModules will be called and control is returned to the caller.

OPTIONAL

This LoginModule is OPTIONAL as per the JAAS specification. The LoginModule is not required to succeed.

true

`libraryRef`

A reference to the ID of the shared library configuration.

Configuration ID of type library (string).

false

`library`

A reference to the ID of the shared library configuration.

Element of type library.

false

`options`

optionsRef.desc

false

jdbcDriver

Identifies a JDBC driver. PID is com.ibm.ws.jdbc.jdbcDriver.

`libraryRef`

Identifies JDBC driver JARs and native files.

Configuration ID of type library (string).

false

`javax.sql.XADataSource`

JDBC driver implementation of javax.sql.XADataSource.

string

false

`javax.sql.ConnectionPoolDataSource`

JDBC driver implementation of javax.sql.ConnectionPoolDataSource.

string

false

javax.sql.DataSource
JDBC driver implementation of javax.sql.DataSource.

string

false

library
Identifies JDBC driver JARs and native files.

Element of type library.

false

jndiEntry

jndi.entry.service.desc. PID is com.ibm.ws.jndi.internal.JNDIEntry.

jndiName
The JNDI name to use for this entry.

string

true

value
The JNDI value to associate with the name.

string

true

jpa

Configuration properties for the Java Persistence API container. PID is com.ibm.ws.jpacomponent.

defaultJtaDataSourceJndiName
Default Java™ Transaction API (JTA) data source JNDI name to be used by applications running in this server. By default, data sources are JTA. Only data sources that are transactional are allowed in this field.

string

false

defaultNonJtaDataSourceJndiName
Default non-transactional data source JNDI name to be used by applications running in this server. Only data sources that have been marked as non-transactional are allowed in this field.

string

false

defaultPersistenceProvider
Default persistence provider.

string

com.ibm.websphere.persistence.PersistenceProviderImpl

false

entityManagerPoolCapacity
EntityManager pool capacity per PersistenceContext reference. The minimum is 0 and the maximum is 500.

int

-1

false

excludedApplication
An application to be excluded from JPA processing.

string

false

jspEngine

JSP 2.2 configuration. PID is com.ibm.ws.jsp.2.2.

disableJspRuntimeCompilation
Disable compilation of JSPs at runtime.

boolean

false

false

extendedDocumentRoot
Directory that the JSP engine will search for additional JSP files to serve.

string

false

jdkSourceLevel
Default Java source level for JSPs compiled by the JSP engine.

string

15

13

14

15

16

false

keepGenerated
Keep Java source files generated for JSPs.

boolean

false

false

useInMemory
Generate Java source and classes in memory (without writing to disk).

boolean

false

false

recompileJspOnRestart
Recompile JSPs after an application is restarted. JSPs are recompiled on first access.

boolean

false

false

useImplicitTagLibs

Allow JSPs to use jsx and tsx tag libs.

boolean

true

false

disableResourceInjection

Disable injection of resources into JSPs.

boolean

false

false

keyStore

A repository of security certificates used for SSL encryption. PID is com.ibm.ws.ssl.keystore.

password

The password used to load the keystore file. The value can be stored in clear text or encoded form. Use the securityUtility tool to encode the password.

password (string)

false

location

An absolute or relative path to the keystore file. If a relative path is provided, the server will attempt to locate the file in the ${server.config.dir}/resources/security directory. Use the keystore file for a file-based keystore, the keyring name for SAF keyrings, or the device configuration file for hardware cryptography devices. In the SSL minimal configuration, the location of the file is assumed to be ${server.config.dir}/resources/security/key.jks.

string

${server.output.dir}/resources/security/key.jks

false

type

A keystore type supported by the target SDK.

string

jks

false

ldapRegistry

Configuration properties for an LDAP user registry. PID is com.ibm.ws.security.registry.ldap.config.

host

Address of the LDAP server in the form of a IP address or a domain name service (DNS) name.

string

true

port

Port number of the LDAP server.

int

true

`baseDN`

Base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches in the directory service.

string

true

`ldapType`

Type of LDAP server to which a connection will be established.

string

Microsoft Active Directory
actived

Custom

IBM Lotus® Domino
domino50

Novell eDirectory
edirectory

IBM Tivoli Directory Server
ibm_dir_server

Sun Java System Directory Server
iplanet

Netscape Directory Server
netscape

IBM SecureWay Directory Server
secureway

true

`realm`

The realm name represents the user registry.

string

LdapRegistry

false

`bindDN`

Distinguished name (DN) for the application server, which is used to bind to the directory service.

string

false

`bindPassword`

Password for the bind DN. The value can be stored in clear text or encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool with the encode option.

password (string)

false

`ignoreCase`

Perform a case-insensitive authentication check.

boolean

true

false

recursiveSearch
Performs a nested group search. Select this option only if the LDAP server does not support recursive server-side searches.

boolean

false

false

reuseConnection
Requests the application server to reuse the LDAP server connection.

boolean

true

false

sslEnabled
Indicates whether an SSL connection should be made to the LDAP server.

boolean

false

false

sslRef
ID of the SSL configuration to be used to connect to the SSL-enabled LDAP server.

string

false

searchTimeout
Maximum time for an LDAP server to respond before a request is canceled. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

2m

false

activedFiltersRef
Specifies the list of default Active Directory LDAP filters.

Configuration ID of type activedLdapFilterProperties (string).

false

customFiltersRef
Specifies the list of default Custom LDAP filters.

Configuration ID of type customLdapFilterProperties (string).

false

domino50FiltersRef
Specifies the list of default Domino LDAP filters.

Configuration ID of type domino50LdapFilterProperties (string).

false

edirectoryFiltersRef
Specifies the list of EDirectory LDAP filters.

Configuration ID of type edirectoryLdapFilterProperties (string).

false

idsFiltersRef
Specifies the list of default IBM Directory Server LDAP filters.

Configuration ID of type idsLdapFilterProperties (string).

false

iplanetFiltersRef
Specifies the list of default IPlanet LDAP filters.

Configuration ID of type iplanetLdapFilterProperties (string).

false

netscapeFiltersRef
Specifies the list of default Netscape LDAP filters.

Configuration ID of type netscapeLdapFilterProperties (string).

false

securewayFiltersRef
Specifies the list of default SecureWay LDAP filters.

Configuration ID of type securewayLdapFilterProperties (string).

false

failoverServers
List of LDAP failover servers.

List of LDAP failover servers.

name
failoverServers.name.desc

string

false

server
Configuration properties for LDAP failover server.

Configuration properties for LDAP failover server.

host
LDAP server host name, which can be either an IP address or a domain name service (DNS) name.

string

true

port
LDAP failover server port.

int

true

false

false

activedFilters

Specifies the list of default Active Directory LDAP filters.

Element of type activedLdapFilterProperties.

false

customFilters

Specifies the list of default Custom LDAP filters.

Element of type customLdapFilterProperties.

false

domino50Filters

Specifies the list of default Domino LDAP filters.

Element of type domino50LdapFilterProperties.

false

edirectoryFilters

Specifies the list of EDirectory LDAP filters.

Element of type edirectoryLdapFilterProperties.

false

idsFilters

Specifies the list of default IBM Directory Server LDAP filters.

Element of type idsLdapFilterProperties.

false

iplanetFilters

Specifies the list of default IPlanet LDAP filters.

Element of type iplanetLdapFilterProperties.

false

netscapeFilters

Specifies the list of default Netscape LDAP filters.

Element of type netscapeLdapFilterProperties.

false

securewayFilters

Specifies the list of default SecureWay LDAP filters.

Element of type securewayLdapFilterProperties.

false

library

Shared Library. PID is com.ibm.ws.classloading.sharedlibrary.

name

Name of shared library for administrators

string

false

description

Description of shared library for administrators

string

false

`filesetRef`

Id of referenced Fileset

List of configuration IDs of type fileset (comma-separated string).

false

`apiTypeVisibility`

The types of API package this library's class loader will be able to see, as a comma-separated list of any combination of the following: spec, ibm-api, ibm-spi, third-party.

string

spec,ibm-api

false

`fileset`

Id of referenced Fileset

Element of type fileset.

false

logging

Controls the output of log and trace messages. PID is com.ibm.ws.logging.

`maxFileSize`

Maximum size of a log file, in megabytes, before being rolled over; a value of 0 means no limit.

com.ibm.ws.logging.max.file.size

int

20

false

`maxFiles`

Maximum number of log files that will be kept, before the oldest file is removed; a value of 0 means no limit.

com.ibm.ws.logging.max.files

int

2

false

`logDirectory`

Location to which all log files will be written. The default value is ${server.output.dir}/logs.

com.ibm.ws.logging.log.directory

string

${server.output.dir}/logs

false

`messageFileName`

Name of the file to which message output will be written relative to the configured log directory. The default value is messages.log.

com.ibm.ws.logging.message.file.name

string

messages.log

false

traceFileName

Name of the file to which trace output will be written relative to the configured log directory. The default value is trace.log.

com.ibm.ws.logging.trace.file.name

string

trace.log

false

traceSpecification

A trace specification that conforms to the trace specification grammar and specifies the initial state for various trace components. An empty value is allowed and treated as 'disable all trace'. Any component that is not specified is initialized to a default state of disabled.

com.ibm.ws.logging.trace.specification

string

*=info=enabled

false

traceFormat

This format is used for the trace log.

com.ibm.ws.logging.trace.format

string

ENHANCED

BASIC

Use the basic trace format.

ENHANCED

Use the enhanced basic trace format.

ADVANCED

Use the advanced trace format.

false

consoleLogLevel

The logging level used to filter messages written to system streams. The default value is audit.

com.ibm.ws.logging.console.log.level

string

AUDIT

INFO Info, audit, and warning messages will be written to the system output stream. Error messages will be written to the system error stream.

AUDIT

Audit and warning messages will be written to the system output stream. Error messages will be written to the system error stream.

WARNING

Warning messages will be written to the system output stream. Error messages will be written to the system error stream.

ERROR
Error messages will be written to the system error stream.

OFF

false

ltpa

Lightweight Third Party Authentication (LTPA) token configuration. PID is com.ibm.ws.security.token.ltpa.LTPAConfiguration.

`keysFileName`
Path of the file containing the token keys.

string

${server.config.dir}/resources/security/ltpa.keys

false

`keysPassword`
Password for the token keys. The value can be stored in clear text or encoded form. It is recommended to encode the password, use the securityUtility tool with the encode option.

password (string)

{xor}CDo9Hgw=

false

`expiration`
Amount of time after which a token expires in minutes.

long

120

false

mimeTypes

Definition of mime types shared by all http virtual hosts. PID is com.ibm.ws.http.mimetype.

`type`
Definition of mime type as id=value. Use the extension as the id, and the associated type as the value.

string

false

monitor

Configuration for Monitoring. PID is com.ibm.ws.monitor.internal.MonitoringFrameworkExtender.

`enableTraditionalPMI`
Internal Property to enable or disable Traditional PMI

boolean

false

false

nativeTransactionManager

Configures the RRS Native Transaction Manager. PID is com.ibm.ws.zos.tx.

`shutdownTimeout`
The amount of time to wait before allowing the native context manager to stop if one or more native contexts are in-use. A native context is considered in-use if it contains an RRS unit of recovery with

one or more interests. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

15s

false

`resourceManagerNamePrefix`

The resource manager prefix to be used as part of the server generated resource manager name.

string

DEFAULT

false

netscapeLdapFilterProperties

Specifies the list of default Netscape LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.netscape.

`userFilter`

An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=inetOrgPerson))

true

`groupFilter`

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))

true

`userIdMap`

An LDAP filter that maps the name of a user to an LDAP entry.

string

inetOrgPerson:uid

true

`groupIdMap`

groupId.desc

string

*:cn

true

`groupMemberIdMap`

An LDAP filter that identifies user to group memberships.

string

groupOfNames:member;groupOfUniqueNames:uniqueMember

true

pluginConfiguration

Generate plugin configuration. PID is com.ibm.ws.generatePluginConfig.

pluginInstallRoot
Web container plugin installation position in file system

string

.

false

webserverPort
Web server HTTP port

string

80

false

webserverSecurePort
Web server HTTPS port

string

443

false

sslKeyringLocation
Location of SSL keyring

string

PATH/FILE

false

sslStashfileLocation
Location of SSL stashfile

string

PATH/FILE

false

sslCertlabel
SSL cert label

string

REPLACE

false

properties

List of JDBC vendor properties for the data source. For example, databaseName="dbname" serverName="localhost" portNumber="50000". PID is com.ibm.ws.jdbc.dataSource.properties, and it is the child of complex type “dataSource”.

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

false

portNumber
Port on which to obtain database connections.

int

false

URL
URL for connecting to the database.

string

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

properties.datadirect.sqlserver

Data source properties for the DataDirect Connect for JDBC driver for Microsoft SQL Server. PID is com.ibm.ws.jdbc.dataSource.properties.datadirect.sqlserver, and it is the child of complex type "dataSource".

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

1433

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

`accountingInfo`
JDBC driver property: accountingInfo.

string

false

`alternateServers`
JDBC driver property: alternateServers.

string

false

`alwaysReportTriggerResults`
JDBC driver property: alwaysReportTriggerResults.

boolean

false

`applicationName`
JDBC driver property: applicationName.

string

false

`authenticationMethod`
JDBC driver property: authenticationMethod.

string

auto

kerberos

ntlm

userIdPassword

false

`bulkLoadBatchSize`
JDBC driver property: bulkLoadBatchSize.

long

false

`bulkLoadOptions`
JDBC driver property: bulkLoadOptions.

long

false

`clientHostName`
JDBC driver property: clientHostName.

string

false

`clientUser`
JDBC driver property: clientUser.

string

false

`codePageOverride`
JDBC driver property: codePageOverride.

string

false

`connectionRetryCount`
JDBC driver property: connectionRetryCount.

int

false

`connectionRetryDelay`
JDBC driver property: connectionRetryDelay. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

`convertNull`
JDBC driver property: convertNull.

int

false

`dateTimeInputParameterType`
JDBC driver property: dateTimeInputParameterType.

string

auto

dateTime

dateTimeOffset

false

`dateTimeOutputParameterType`
JDBC driver property: dateTimeOutputParameterType.

string

auto

dateTime

dateTimeOffset

false

`describeInputParameters`
JDBC driver property: describeInputParameters.

string

describeAll

describeIfDateTime

describeIfString

noDescribe

false

`describeOutputParameters`

JDBC driver property: describeOutputParameters.

string

describeAll

describeIfDateTime

describeIfString

noDescribe

false

`enableBulkLoad`

JDBC driver property: enableBulkLoad.

boolean

false

`enableCancelTimeout`

JDBC driver property: enableCancelTimeout.

boolean

false

`encryptionMethod`

JDBC driver property: encryptionMethod.

string

noEncryption

loginSSL

requestSSL

SSL

false

`failoverGranularity`

JDBC driver property: failoverGranularity.

string

atomic

atomicWithRepositioning

disableIntegrityCheck

nonAtomic

false

`failoverMode`

JDBC driver property: failoverMode.

string

connect

extended

select

false

`failoverPreconnect`
JDBC driver property: failoverPreconnect.

boolean

false

`hostNameInCertificate`
JDBC driver property: hostNameInCertificate.

string

false

`initializationString`
JDBC driver property: initializationString.

string

false

`insensitiveResultSetBufferSize`
JDBC driver property: insensitiveResultSetBufferSize.

int

false

`javaDoubleToString`
JDBC driver property: javaDoubleToString.

boolean

false

`JDBCBehavior`
JDBC driver property: JDBCBehavior. Values are: 0 (JDBC 4.0) or 1 (JDBC 3.0).

int

0

0 JDBC 4.0

1 JDBC 3.0

false

`loadBalancing`
JDBC driver property: loadBalancing.

boolean

false

`loginTimeout`
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

`longDataCacheSize`
JDBC driver property: longDataCacheSize.

int

false

netAddress
JDBC driver property: netAddress.

string

false

packetSize
JDBC driver property: packetSize.

int

false

queryTimeout
queryTimeout.datadirect.sqlserver.desc. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

resultsetMetaDataOptions
JDBC driver property: resultsetMetaDataOptions.

int

false

selectMethod
JDBC driver property: selectMethod.

string

cursor

direct

false

snapshotSerializable
JDBC driver property: snapshotSerializable.

boolean

false

spyAttributes
JDBC driver property: spyAttributes.

string

false

stringInputParameterType
JDBC driver property: stringInputParameterType.

string

varchar

nvarchar

varchar

false

stringOutputParameterType

JDBC driver property: stringOutputParameterType.

string

varchar

nvarchar

varchar

false

suppressConnectionWarnings

JDBC driver property: suppressConnectionWarnings.

boolean

false

transactionMode

JDBC driver property: transactionMode.

string

explicit

implicit

false

truncateFractionalSeconds

JDBC driver property: truncateFractionalSeconds.

boolean

false

trustStore

JDBC driver property: trustStore.

string

false

trustStorePassword

JDBC driver property: trustStorePassword.

password (string)

false

useServerSideUpdatableCursors

JDBC driver property: useServerSideUpdatableCursors.

boolean

false

validateServerCertificate

JDBC driver property: validateServerCertificate.

boolean

false

XATransactionGroup

JDBC driver property: XATransactionGroup.

string

false

`XMLDescribeType`
JDBC driver property: XMLDescribeType.

string

longvarbinary

longvarchar

false

properties.db2.i.native

Data source properties for the IBM DB2 for i Native JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.db2.i.native, and it is the child of complex type “dataSource”.

`databaseName`
JDBC driver property: databaseName.

string

*LOCAL

false

`user`
Database user name.

string

false

`password`
Password for database user.

password (string)

false

`access`
JDBC driver property: access.

string

all

all

read call

read only

false

`autoCommit`
JDBC driver property: autoCommit.

boolean

true

false

`batchStyle`
JDBC driver property: batchStyle.

string

2.0

2.0

2.1

false

behaviorOverride
JDBC driver property: behaviorOverride.

int

false

blockSize
JDBC driver property: blockSize.

int

32

0

8

16

32

64

128

256

512

false

cursorHold
JDBC driver property: cursorHold.

boolean

false

false

cursorSensitivity
JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1 (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

string

asensitive

asensitive

sensitive

false

dataTruncation
JDBC driver property: dataTruncation.

string

true

false

dateFormat
JDBC driver property: dateFormat.

string

dmy

eur

mdy

iso

jis

julian

usa

ymd

false

`dateSeparator`
JDBC driver property: dateSeparator.

string

/ The forward slash character (/).

- The dash character (-).

. The period character (.).

, The comma character (,).

b The character b

false

`decimalSeparator`
JDBC driver property: decimalSeparator.

string

. The period character (.).

, The comma character (,).

false

`directMap`
JDBC driver property: directMap.

boolean

false

false

`doEscapeProcessing`
JDBC driver property: doEscapeProcessing.

boolean

true

false

`fullErrors`
JDBC driver property: fullErrors.

boolean

false

`libraries`
JDBC driver property: libraries.

string

false

`lobThreshold`
JDBC driver property: lobThreshold.

int

0

false

`lockTimeout`
JDBC driver property: lockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

`loginTimeout`
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

`maximumPrecision`
JDBC driver property: maximumPrecision.

int

31

31

63

false

`maximumScale`
JDBC driver property: maximumScale.

int

31

false

`minimumDivideScale`
JDBC driver property: minimumDivideScale.

int

0

false

`networkProtocol`
JDBC driver property: networkProtocol.

int

false

portNumber
Port on which to obtain database connections.

int

false

prefetch
JDBC driver property: prefetch.

boolean

true

false

queryOptimizeGoal
JDBC driver property: queryOptimizeGoal. Values are: 1 (*FIRSTIO) or 2 (*ALLIO).

string

2

1	*FIRSTIO
2	*ALLIO

false

reuseObjects
JDBC driver property: reuseObjects.

boolean

true

false

serverName
Server where the database is running.

string

false

serverTraceCategories
JDBC driver property: serverTraceCategories.

int

0

false

systemNaming
JDBC driver property: systemNaming.

boolean

false

false

timeFormat
JDBC driver property: timeFormat.

string

eur

hms

iso

jis

usa

false

`timeSeparator`

JDBC driver property: timeSeparator.

string

: The colon character (:).

. The period character (.).

, The comma character (,).

b The character b

false

`trace`

JDBC driver property: trace.

boolean

false

`transactionTimeout`

JDBC driver property: transactionTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

`translateBinary`

JDBC driver property: translateBinary.

boolean

false

false

`translateHex`

JDBC driver property: translateHex.

string

character

binary

character

false

`useBlockInsert`

JDBC driver property: useBlockInsert.

boolean

false

false

properties.db2.i.toolbox

Data source properties for the IBM DB2 for i Toolbox JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.db2.i.toolbox, and it is the child of complex type "dataSource".

`serverName`

Server where the database is running.

string

true

`databaseName`

JDBC driver property: databaseName.

string

false

`user`

Database user name.

string

false

`password`

Password for database user.

password (string)

false

`access`

JDBC driver property: access.

string

all

all

read call

read only

false

`behaviorOverride`

JDBC driver property: behaviorOverride.

int

false

`bidiImplicitReordering`

JDBC driver property: bidiImplicitReordering.

boolean

true

false

`bidiNumericOrdering`

JDBC driver property: bidiNumericOrdering.

boolean

false

false

`bidiStringType`
JDBC driver property: bidiStringType.

int

false

`bigDecimal`
JDBC driver property: bigDecimal.

boolean

true

false

`blockCriteria`
JDBC driver property: blockCriteria. Values are: 0 (no record blocking), 1 (block if FOR FETCH ONLY is specified), 2 (block if FOR UPDATE is specified).

int

2

0

1

2

false

`blockSize`
JDBC driver property: blockSize.

int

32

0

8

16

32

64

128

256

512

false

`cursorHold`
JDBC driver property: cursorHold.

boolean

false

false

`cursorSensitivity`

JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1 (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

string

asensitive

asensitive

insensitive

sensitive

false

`dataCompression`

JDBC driver property: dataCompression.

boolean

true

false

`dataTruncation`

JDBC driver property: dataTruncation.

boolean

true

false

`dateFormat`

JDBC driver property: dateFormat.

string

dmy

eur

mdy

iso

jis

julian

usa

ymd

false

`dateSeparator`

JDBC driver property: dateSeparator.

string

/ The forward slash character (/).

- The dash character (-).

. The period character (.).

, The comma character (,).

The space character ( ).

false

decimalSeparator
JDBC driver property: decimalSeparator.

string

. The period character (.).

, The comma character (,).

false

driver
JDBC driver property: driver.

string

toolbox

native

toolbox

false

errors
JDBC driver property: errors.

string

basic

basic

full

false

extendedDynamic
JDBC driver property: extendedDynamic.

boolean

false

false

extendedMetaData
JDBC driver property: extendedMetaData.

boolean

false

false

fullOpen
JDBC driver property: fullOpen.

boolean

false

false

holdInputLocators
JDBC driver property: holdInputLocators.

boolean

true

false

holdStatements

JDBC driver property: holdStatements.

boolean

false

false

isolationLevelSwitchingSupport

JDBC driver property: isolationLevelSwitchingSupport.

boolean

false

false

keepAlive

JDBC driver property: keepAlive.

boolean

false

lazyClose

JDBC driver property: lazyClose.

boolean

false

false

libraries

JDBC driver property: libraries.

string

false

lobThreshold

JDBC driver property: lobThreshold.

int

0

false

loginTimeout

JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

maximumPrecision

JDBC driver property: maximumPrecision.

int

31

31

63 64

false

maximumScale
JDBC driver property: maximumScale.

int

31

false

metaDataSource
JDBC driver property: metaDataSource.

int

1

false

minimumDivideScale
JDBC driver property: minimumDivideScale.

int

0

false

naming
JDBC driver property: naming.

string

sql

sql

system

false

package
JDBC driver property: package.

string

false

packageAdd
JDBC driver property: packageAdd.

boolean

true

false

packageCCSID
JDBC driver property: packageCCSID. Values are: 1200 (UCS-2) or 13488 (UTF-16).

int

13488

1200 1200 (UCS-2)

13488 13488 (UTF-16)

false

packageCache
JDBC driver property: packageCache.

boolean

false

false

packageCriteria
JDBC driver property: packageCriteria.

string

default

default

select

false

packageError
JDBC driver property: packageError.

string

warning

exception

warning

none

false

packageLibrary
JDBC driver property: packageLibrary.

string

QGPL

false

prefetch
JDBC driver property: prefetch.

boolean

true

false

prompt
JDBC driver property: prompt.

boolean

false

false

proxyServer
JDBC driver property: proxyServer.

string

false

qaqqiniLibrary

JDBC driver property: qaqqiniLibrary.

string

false

queryOptimizeGoal

JDBC driver property: queryOptimizeGoal. Values are: 1 (*FIRSTIO) or 2 (*ALLIO).

int

0

false

receiveBufferSize

JDBC driver property: receiveBufferSize.

int

false

remarks

JDBC driver property: remarks.

string

system

sql

system

false

rollbackCursorHold

JDBC driver property: rollbackCursorHold.

boolean

false

false

savePasswordWhenSerialized

JDBC driver property: savePasswordWhenSerialized.

boolean

false

false

secondaryUrl

JDBC driver property: secondaryUrl.

string

false

secure

JDBC driver property: secure.

boolean

false

false

sendBufferSize

JDBC driver property: sendBufferSize.

int

false

serverTraceCategories

JDBC driver property: serverTraceCategories.

int

0

false

soLinger

JDBC driver property: soLinger. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

soTimeout

JDBC driver property: soTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

sort

JDBC driver property: sort.

string

hex

hex

language

table

false

sortLanguage

JDBC driver property: sortLanguage.

string

false

sortTable

JDBC driver property: sortTable.

string

false

sortWeight

JDBC driver property: sortWeight.

string

shared

unqiue
unique

false

tcpNoDelay
JDBC driver property: tcpNoDelay.

boolean

false

threadUsed
JDBC driver property: threadUsed.

boolean

true

false

timeFormat
JDBC driver property: timeFormat.

string

eur

hms

iso

jis

usa

false

timeSeparator
JDBC driver property: timeSeparator.

string

: The colon character (:).

. The period character (.).

, The comma character (,).

The space character ( ).

false

toolboxTrace
JDBC driver property: toolboxTrace.

string

none

datastream

diagnostic

error

information

warning

conversion

proxy

pcml

jdbc

all

thread

false

`trace`
JDBC driver property: trace.

boolean

false

`translateBinary`
JDBC driver property: translateBinary.

boolean

false

false

`translateBoolean`
JDBC driver property: translateBoolean.

boolean

true

false

`translateHex`
JDBC driver property: translateHex.

string

character

binary

character

false

`trueAutoCommit`
JDBC driver property: trueAutoCommit.

boolean

false

false

`xaLooselyCoupledSupport`
JDBC driver property: xaLooselyCoupledSupport.

int

0

false

properties.db2.jcc

Data source properties for the IBM Data Server Driver for JDBC and SQLJ for DB2. PID is com.ibm.ws.jdbc.dataSource.properties.db2.jcc, and it is the child of complex type “dataSource”.

driverType
JDBC driver property: driverType.

int

4

2 Type 2 JDBC driver.

4 Type 4 JDBC driver.

false

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

50000

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

blockingReadConnectionTimeout
JDBC driver property: blockingReadConnectionTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

clientAccountingInformation
JDBC driver property: clientAccountingInformation.

string

false

clientApplicationInformation
JDBC driver property: clientApplicationInformation.

string

false

clientRerouteServerListJNDIName
JDBC driver property: clientRerouteServerListJNDIName.

string

false

clientUser
JDBC driver property: clientUser.

string

false

clientWorkstation
JDBC driver property: clientWorkstation.

string

false

currentFunctionPath
JDBC driver property: currentFunctionPath.

string

false

currentLockTimeout
JDBC driver property: currentLockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

currentPackagePath
JDBC driver property: currentPackagePath.

string

false

currentPackageSet
JDBC driver property: currentPackageSet.

string

false

currentSQLID
JDBC driver property: currentSQLID.

string

false

currentSchema
JDBC driver property: currentSchema.

string

false

cursorSensitivity
JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1 (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

int

0 TYPE_SCROLL_SENSITIVE_STATIC

1 TYPE_SCROLL_SENSITIVE_DYNAMIC

2 TYPE_SCROLL_ASENSITIVE

false

deferPrepares
JDBC driver property: deferPrepares.

boolean

true

false

enableClientAffinitiesList
JDBC driver property: enableClientAffinitiesList. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

enableNamedParameterMarkers
JDBC driver property: enableNamedParameterMarkers. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

enableSeamlessFailover
JDBC driver property: enableSeamlessFailover. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

enableSysplexWLB
JDBC driver property: enableSysplexWLB.

boolean

false

fetchSize
JDBC driver property: fetchSize.

int

false

fullyMaterializeInputStreams

JDBC driver property: fullyMaterializeInputStreams.

boolean

false

fullyMaterializeLobData

JDBC driver property: fullyMaterializeLobData.

boolean

false

keepDynamic

JDBC driver property: keepDynamic.

int

false

kerberosServerPrincipal

JDBC driver property: kerberosServerPrincipal.

string

false

loginTimeout

JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

maxRetriesForClientReroute

JDBC driver property: maxRetriesForClientReroute.

int

false

queryCloseImplicit

JDBC driver property: queryCloseImplicit. Values are: 1 (QUERY_CLOSE_IMPLICIT_YES) or 2 (QUERY_CLOSE_IMPLICIT_NO).

int

1 QUERY_CLOSE_IMPLICIT_YES

2 QUERY_CLOSE_IMPLICIT_NO

false

queryDataSize

JDBC driver property: queryDataSize.

int

false

readOnly

JDBC driver property: readOnly.

boolean

false

`resultSetHoldability`

JDBC driver property: resultSetHoldability. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1	HOLD_CURSORS_OVER_COMMIT
2	CLOSE_CURSORS_AT_COMMIT

false

`resultSetHoldabilityForCatalogQueries`

JDBC driver property: resultSetHoldabilityForCatalogQueries. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1	HOLD_CURSORS_OVER_COMMIT
2	CLOSE_CURSORS_AT_COMMIT

false

`retrieveMessagesFromServerOnGetMessage`

JDBC driver property: retrieveMessagesFromServerOnGetMessage.

boolean

true

false

`retryIntervalForClientReroute`

JDBC driver property: retryIntervalForClientReroute. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

`securityMechanism`

JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4 (USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 9 (ENCRYPTED_USER_AND_PASSWORD_SECURITY), 11 (KERBEROS_SECURITY), 12 (ENCRYPTED_USER_AND_DATA_SECURITY"), (ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY"), 15 (PLUGIN_SECURITY), 16 (ENCRYPTED_USER_ONLY_SECURITY).

int

3	CLEAR_TEXT_PASSWORD_SECURITY
4	USER_ONLY_SECURITY
7	ENCRYPTED_PASSWORD_SECURITY
9	ENCRYPTED_USER_AND_PASSWORD_SECURITY
11	KERBEROS_SECURITY
12	ENCRYPTED_USER_AND_DATA_SECURITY
13	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
15	PLUGIN_SECURITY
16	ENCRYPTED_USER_ONLY_SECURITY

false

sendDataAsIs
JDBC driver property: sendDataAsIs.

boolean

false

sqljEnableClassLoaderSpecificProfiles
JDBC driver property: sqljEnableClassLoaderSpecificProfiles.

boolean

false

sslConnection
JDBC driver property: sslConnection.

boolean

false

streamBufferSize
JDBC driver property: streamBufferSize.

int

false

sysSchema
JDBC driver property: sysSchema.

string

false

traceDirectory
JDBC driver property: traceDirectory.

string

false

traceFile
JDBC driver property: traceFile.

string

false

traceFileAppend
JDBC driver property: traceFileAppend.

boolean

false

traceLevel
Bitwise combination of the following constant values: TRACE_NONE=0, TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2, TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32, TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128, TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_SQLJ=1024, TRACE_META_CALLS=8192, TRACE_DATASOURCE_CALLS=16384, TRACE_LARGE_OBJECT_CALLS=32768, TRACE_SYSTEM_MONITOR=131072, TRACE_TRACEPOINTS=262144, TRACE_ALL=-1.

int

0

false

useCachedCursor
JDBC driver property: useCachedCursor.

boolean

false

useJDBC4ColumnNameAndLabelSemantics
JDBC driver property: useJDBC4ColumnNameAndLabelSemantics. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

useTransactionRedirect
JDBC driver property: useTransactionRedirect.

boolean

false

xaNetworkOptimization
JDBC driver property: xaNetworkOptimization.

boolean

false

properties.derby.client

Data source properties for Derby Network Client JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.derby.client, and it is the child of complex type "dataSource".

createDatabase
JDBC driver property: createDatabase.

string

create When the first connection is established, automatically create the database if it doesn't exist.

false Do not automatically create the database.

false

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

1527

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

connectionAttributes
JDBC driver property: connectionAttributes.

string

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

retrieveMessageText
JDBC driver property: retrieveMessageText.

boolean

true

false

securityMechanism
JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4 (USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 8 (STRONG_PASSWORD_SUBSTITUTE_SECURITY), 9 (ENCRYPTED_USER_AND_PASSWORD_SECURITY).

short

3

3	CLEAR_TEXT_PASSWORD_SECURITY
4	USER_ONLY_SECURITY
7	ENCRYPTED_PASSWORD_SECURITY
8	STRONG_PASSWORD_SUBSTITUTE_SECURITY
9	ENCRYPTED_USER_AND_PASSWORD_SECURITY

false

shutdownDatabase
JDBC driver property: shutdownDatabase.

string

shutdown
Shut down the database when a connection is attempted.

false Do not shut down the database.

false

`ssl`
JDBC driver property: ssl.

string

basic

peerAuthentication

off

false

`traceDirectory`
JDBC driver property: traceDirectory.

string

false

`traceFile`
JDBC driver property: traceFile.

string

false

`traceFileAppend`
JDBC driver property: traceFileAppend.

boolean

false

`traceLevel`
Bitwise combination of the following constant values: TRACE_NONE=0, TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2, TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32, TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128, TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_XA_CALLS=2048, TRACE_ALL=-1.

int

false

properties.derby.embedded

Data source properties for Derby Embedded JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.derby.embedded, and it is the child of complex type “dataSource”.

`createDatabase`
JDBC driver property: createDatabase.

string

create When the first connection is established, automatically create the database if it doesn't exist.

false Do not automatically create the database.

false

databaseName
JDBC driver property: databaseName.

string

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

connectionAttributes
JDBC driver property: connectionAttributes.

string

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

shutdownDatabase
JDBC driver property: shutdownDatabase.

string

shutdown
Shut down the database when a connection is attempted.

false Do not shut down the database.

false

properties.informix

Data source properties for the Informix JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.informix, and it is the child of complex type “dataSource”.

databaseName
JDBC driver property: databaseName.

string

false

ifxIFXHOST
JDBC driver property: ifxIFXHOST.

string

localhost

false

serverName

Server where the database is running.

string

false

portNumber

Port on which to obtain database connections.

int

1526

false

user

Database user name.

string

false

password

Password for database user.

password (string)

false

ifxCLIENT_LOCALE

JDBC driver property: ifxCLIENT_LOCALE.

string

false

ifxCPMAgeLimit

JDBC driver property: ifxCPMAgeLimit. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

ifxCPMInitPoolSize

JDBC driver property: ifxCPMInitPoolSize.

int

false

ifxCPMMaxConnections

JDBC driver property: ifxCPMMaxConnections.

int

false

ifxCPMMaxPoolSize

JDBC driver property: ifxCPMMaxPoolSize.

int

false

ifxCPMMinAgeLimit

JDBC driver property: ifxCPMMinAgeLimit. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

ifxCPMMinPoolSize
JDBC driver property: ifxCPMMinPoolSize.

int

false

ifxCPMServiceInterval
JDBC driver property: ifxCPMServiceInterval. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

ifxDBANSIWARN
JDBC driver property: ifxDBANSIWARN.

boolean

false

ifxDBCENTURY
JDBC driver property: ifxDBCENTURY.

string

false

ifxDBDATE
JDBC driver property: ifxDBDATE.

string

false

ifxDBSPACETEMP
JDBC driver property: ifxDBSPACETEMP.

string

false

ifxDBTEMP
JDBC driver property: ifxDBTEMP.

string

false

ifxDBTIME
JDBC driver property: ifxDBTIME.

string

false

ifxDBUPSPACE
JDBC driver property: ifxDBUPSPACE.

string

false

ifxDB_LOCALE

JDBC driver property: ifxDB_LOCALE.

string

false

ifxDELIMIDENT

JDBC driver property: ifxDELIMIDENT.

boolean

false

ifxENABLE_TYPE_CACHE

JDBC driver property: ifxENABLE_TYPE_CACHE.

boolean

false

ifxFET_BUF_SIZE

JDBC driver property: ifxFET_BUF_SIZE.

int

false

ifxGL_DATE

JDBC driver property: ifxGL_DATE.

string

false

ifxGL_DATETIME

JDBC driver property: ifxGL_DATETIME.

string

false

ifxIFX_AUTOFREE

JDBC driver property: ifxIFX_AUTOFREE.

boolean

false

ifxIFX_DIRECTIVES

JDBC driver property: ifxIFX_DIRECTIVES.

string

false

ifxIFX_LOCK_MODE_WAIT

ifxIFX_LOCK_MODE_WAIT.desc. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

2s

false

ifxIFX_SOC_TIMEOUT

JDBC driver property: ifxIFX_SOC_TIMEOUT. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

ifxIFX_USEPUT

JDBC driver property: ifxIFX_USEPUT.

boolean

false

ifxIFX_USE_STRENC

JDBC driver property: ifxIFX_USE_STRENC.

boolean

false

ifxIFX_XASPEC

JDBC driver property: ifxIFX_XASPEC.

string

y

false

ifxINFORMIXCONRETRY

JDBC driver property: ifxINFORMIXCONRETRY.

int

false

ifxINFORMIXCONTIME

JDBC driver property: ifxINFORMIXCONTIME. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

ifxINFORMIXOPCACHE

JDBC driver property: ifxINFORMIXOPCACHE.

string

false

ifxINFORMIXSTACKSIZE

JDBC driver property: ifxINFORMIXSTACKSIZE.

int

false

ifxJDBCTEMP

JDBC driver property: ifxJDBCTEMP.

string

false

ifxLDAP_IFXBASE
JDBC driver property: ifxLDAP_IFXBASE.

string

false

ifxLDAP_PASSWD
JDBC driver property: ifxLDAP_PASSWD.

string

false

ifxLDAP_URL
JDBC driver property: ifxLDAP_URL.

string

false

ifxLDAP_USER
JDBC driver property: ifxLDAP_USER.

string

false

ifxLOBCACHE
JDBC driver property: ifxLOBCACHE.

int

false

ifxNEWCODESET
JDBC driver property: ifxNEWCODESET.

string

false

ifxNEWLOCALE
JDBC driver property: ifxNEWLOCALE.

string

false

ifxNODEFDAC
JDBC driver property: ifxNODEFDAC.

string

false

ifxOPTCOMPIND
JDBC driver property: ifxOPTCOMPIND.

string

false

ifxOPTOFC
JDBC driver property: ifxOPTOFC.

string

false

ifxOPT_GOAL
JDBC driver property: ifxOPT_GOAL.

string

false

ifxPATH
JDBC driver property: ifxPATH.

string

false

ifxPDQPRIORITY
JDBC driver property: ifxPDQPRIORITY.

string

false

ifxPLCONFIG
JDBC driver property: ifxPLCONFIG.

string

false

ifxPLOAD_LO_PATH
JDBC driver property: ifxPLOAD_LO_PATH.

string

false

ifxPROTOCOLTRACE
JDBC driver property: ifxPROTOCOLTRACE.

int

false

ifxPROTOCOLTRACEFILE
JDBC driver property: ifxPROTOCOLTRACEFILE.

string

false

ifxPROXY
JDBC driver property: ifxPROXY.

string

false

ifxPSORT_DBTEMP
JDBC driver property: ifxPSORT_DBTEMP.

string

false

ifxPSORT_NPROCS
JDBC driver property: ifxPSORT_NPROCS.

boolean

false

`ifxSECURITY`
JDBC driver property: ifxSECURITY.

string

false

`ifxSQLH_FILE`
JDBC driver property: ifxSQLH_FILE.

string

false

`ifxSQLH_LOC`
JDBC driver property: ifxSQLH_LOC.

string

false

`ifxSQLH_TYPE`
JDBC driver property: ifxSQLH_TYPE.

string

false

`ifxSSLCONNECTION`
JDBC driver property: ifxSSLCONNECTION.

string

false

`ifxSTMT_CACHE`
JDBC driver property: ifxSTMT_CACHE.

string

false

`ifxTRACE`
JDBC driver property: ifxTRACE.

int

false

`ifxTRACEFILE`
JDBC driver property: ifxTRACEFILE.

string

false

`ifxTRUSTED_CONTEXT`
JDBC driver property: ifxTRUSTED_CONTEXT.

string

false

`ifxUSEV5SERVER`
JDBC driver property: ifxUSEV5SERVER.

boolean

false

ifxUSE_DTENV
JDBC driver property: ifxUSE_DTENV.

boolean

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

roleName
JDBC driver property: roleName.

string

false

properties.informix.jcc

Data source properties for the IBM Data Server Driver for JDBC and SQLJ for Informix. PID is com.ibm.ws.jdbc.dataSource.properties.informix.jcc, and it is the child of complex type “dataSource”.

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

1526

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

currentLockTimeout
JDBC driver property: currentLockTimeout. Specify a positive integer followed by a unit of time, which

can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

2s

false

DBANSIWARN
JDBC driver property: DBANSIWARN.

boolean

false

DBDATE
JDBC driver property: DBDATE.

string

false

DBPATH
JDBC driver property: DBPATH.

string

false

DBSPACETEMP
JDBC driver property: DBSPACETEMP.

string

false

DBTEMP
JDBC driver property: DBTEMP.

string

false

DBUPSPACE
JDBC driver property: DBUPSPACE.

string

false

DELIMIDENT
JDBC driver property: DELIMIDENT.

boolean

false

deferPrepares
JDBC driver property: deferPrepares.

boolean

false

driverType
JDBC driver property: driverType.

int

4

false

enableNamedParameterMarkers
JDBC driver property: enableNamedParameterMarkers. Values are: 1 (YES) or 2 (NO).

int

false

enableSeamlessFailover
JDBC driver property: enableSeamlessFailover. Values are: 1 (YES) or 2 (NO).

int

false

enableSysplexWLB
JDBC driver property: enableSysplexWLB.

boolean

false

fetchSize
JDBC driver property: fetchSize.

int

false

fullyMaterializeLobData
JDBC driver property: fullyMaterializeLobData.

boolean

false

IFX_DIRECTIVES
JDBC driver property: IFX_DIRECTIVES.

string

ON

OFF

false

IFX_EXTDIRECTIVES
JDBC driver property: IFX_EXTDIRECTIVES.

string

ON

OFF

false

IFX_UPDDESC
JDBC driver property: IFX_UPDDESC.

string

false

IFX_XASTDCOMPLIANCE_XAEND
JDBC driver property: IFX_XASTDCOMPLIANCE_XAEND.

string

0

1

false

INFORMIXOPCACHE

JDBC driver property: INFORMIXOPCACHE.

string

false

INFORMIXSTACKSIZE

JDBC driver property: INFORMIXSTACKSIZE.

string

false

keepDynamic

JDBC driver property: keepDynamic.

int

false

loginTimeout

JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

NODEFDAC

JDBC driver property: NODEFDAC.

string

yes

no

false

OPTCOMPIND

JDBC driver property: OPTCOMPIND.

string

0

1

2

false

OPTOFC

JDBC driver property: OPTOFC.

string

0

1

false

PDQPRIORITY
JDBC driver property: PDQPRIORITY.

string

HIGH

LOW

OFF

false

progressiveStreaming
JDBC driver property: progressiveStreaming. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

PSORT_DBTEMP
JDBC driver property: PSORT_DBTEMP.

string

false

PSORT_NPROCS
JDBC driver property: PSORT_NPROCS.

string

false

queryDataSize
JDBC driver property: queryDataSize.

int

false

resultSetHoldability
JDBC driver property: resultSetHoldability. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1 HOLD_CURSORS_OVER_COMMIT

2 CLOSE_CURSORS_AT_COMMIT

false

resultSetHoldabilityForCatalogQueries
JDBC driver property: resultSetHoldabilityForCatalogQueries. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1 HOLD_CURSORS_OVER_COMMIT

2 CLOSE_CURSORS_AT_COMMIT

false

retrieveMessagesFromServerOnGetMessage
JDBC driver property: retrieveMessagesFromServerOnGetMessage.

boolean

true

false

securityMechanism
JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4 (USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 9 (ENCRYPTED_USER_AND_PASSWORD_SECURITY).

short

3	CLEAR_TEXT_PASSWORD_SECURITY
4	USER_ONLY_SECURITY
7	ENCRYPTED_PASSWORD_SECURITY
9	ENCRYPTED_USER_AND_PASSWORD_SECURITY

false

STMT_CACHE
JDBC driver property: STMT_CACHE.

string

0

1

false

traceDirectory
JDBC driver property: traceDirectory.

string

false

traceFile
JDBC driver property: traceFile.

string

false

traceFileAppend
JDBC driver property: traceFileAppend.

boolean

false

traceLevel
Bitwise combination of the following constant values: TRACE_NONE=0, TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2, TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32, TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128, TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_SQLJ=1024, TRACE_META_CALLS=8192, TRACE_DATASOURCE_CALLS=16384, TRACE_LARGE_OBJECT_CALLS=32768, TRACE_SYSTEM_MONITOR=131072, TRACE_TRACEPOINTS=262144, TRACE_ALL=-1.

int

false

useJDBC4ColumnNameAndLabelSemantics
JDBC driver property: useJDBC4ColumnNameAndLabelSemantics. Values are: 1 (YES) or 2 (NO).

int

false

properties.microsoft.sqlserver

Data source properties for Microsoft SQL Server JDBC Driver. PID is com.ibm.ws.jdbc.dataSource.properties.microsoft.sqlserver, and it is the child of complex type “dataSource”.

databaseName
JDBC driver property: databaseName.

string

false

instanceName
JDBC driver property: instanceName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

1433

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

applicationIntent
JDBC driver property: applicationIntent.

string

ReadOnly

ReadWrite

false

applicationName
JDBC driver property: applicationName.

string

false

authenticationScheme
JDBC driver property: authenticationScheme.

string

JavaKerberos

NativeAuthentication

false

encrypt
JDBC driver property: encrypt.

boolean

false

failoverPartner
JDBC driver property: failoverPartner.

string

false

hostNameInCertificate
JDBC driver property: hostNameInCertificate.

string

false

integratedSecurity
JDBC driver property: integratedSecurity.

boolean

false

lastUpdateCount
JDBC driver property: lastUpdateCount.

boolean

false

lockTimeout
JDBC driver property: lockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

multiSubnetFailover
JDBC driver property: multiSubnetFailover.

boolean

false

packetSize
JDBC driver property: packetSize.

int

false

responseBuffering
JDBC driver property: responseBuffering.

string

adaptive

full

false

selectMethod
JDBC driver property: selectMethod.

string

cursor

direct

false

sendStringParametersAsUnicode
JDBC driver property: sendStringParametersAsUnicode.

boolean

false

false

sendTimeAsDatetime
JDBC driver property: sendTimeAsDatetime.

boolean

false

trustServerCertificate
JDBC driver property: trustServerCertificate.

boolean

false

trustStore
JDBC driver property: trustStore.

string

false

trustStorePassword
JDBC driver property: trustStorePassword.

password (string)

false

URL

URL for connecting to the database. Example: jdbc:sqlserver://localhost:1433;databaseName=myDB.

string

false

workstationID

JDBC driver property: workstationID.

string

false

xopenStates

JDBC driver property: xopenStates.

boolean

false

properties.oracle

Data source properties for Oracle JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.oracle, and it is the child of complex type “dataSource”.

driverType

JDBC driver property: driverType.

string

thin

thin

oci

false

databaseName

JDBC driver property: databaseName.

string

false

serverName

Server where the database is running.

string

localhost

false

portNumber

Port on which to obtain database connections.

int

1521

false

URL

URL for connecting to the database. Examples: jdbc:oracle:thin:@//localhost:1521/sample or jdbc:oracle:oci:@//localhost:1521/sample.

string

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

connectionProperties
JDBC driver property: connectionProperties.

string

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

networkProtocol
JDBC driver property: networkProtocol.

string

false

ONSConfiguration
JDBC driver property: ONSConfiguration.

string

false

serviceName
JDBC driver property: serviceName.

string

false

TNSEntryName
JDBC driver property: TNSEntryName.

string

false

properties.sybase

Data source properties for Sybase JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.sybase, and it is the child of complex type “dataSource”.

databaseName
JDBC driver property: databaseName.

string

true

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

5000

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

connectionProperties
JDBC driver property: connectionProperties.

string

SELECT_OPENS_CURSOR=true

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

networkProtocol
JDBC driver property: networkProtocol.

string

socket

SSL

false

resourceManagerName
JDBC driver property: resourceManagerName.

string

false

`SERVER_INITIATED_TRANSACTIONS`

JDBC driver property: SERVER_INITIATED_TRANSACTIONS.

string

true

false

false

`version`

JDBC driver property: version.

int

false

quickStartSecurity

Simple administrative security configuration. PID is com.ibm.ws.security.quickStartSecurity.

`userName`

Single user defined as part of the quick start security configuration. This user is granted the Administrator role.

string

true

`userPassword`

Password for the single user defined as part of the quick start security configuration. It is recommended that you encode this password. To do so, use the securityUtility tool with the encode option.

password (string)

true

safAuthorization

Controls the operation of the SAF Authorization Service. PID is com.ibm.ws.security.authorization.saf.

`roleMapper`

OSGi component name of the SAF Role Mapper service provider.

string

com.ibm.ws.security.authorization.saf.internal.SAFRoleMapperImpl

false

safCredentials

Controls the operation of the SAF Credentials Service. PID is com.ibm.ws.security.credentials.saf.

`unauthenticatedUser`

SAF user ID of the unauthenticated user.

string

WSGUEST

false

`profilePrefix`

Profile prefix used to specify the SAF APPL-ID when creating SAF credentials and authorizing users against SAF resource profiles.

string

BBGZDFLT

false

safRegistry

Configuration properties for a SAF user registry. PID is com.ibm.ws.security.registry.saf.config.

realm

The realm name represents the user registry.

string

false

safRoleMapper

Defines how to generate SAF EJBROLE resource profile names from application role names. PID is com.ibm.ws.security.authorization.saf.internal.SAFRoleMapperImpl.

profilePattern

Pattern to use for generating EJBROLE resource profile names from application role names.

string

%profilePrefix%.%resource%.%role%

false

toUpperCase

Convert the EJBROLE resource profile name to upper-case.

boolean

false

false

securewayLdapFilterProperties

Specifies the list of default SecureWay LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.secureway.

userFilter

An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=ePerson))

true

groupFilter

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))

true

userIdMap

An LDAP filter that maps the name of a user to an LDAP entry.

string

*:uid

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap
An LDAP filter that identifies user to group memberships.

string

groupOfNames:member;groupOfUniqueNames:uniqueMember

true

ssl

An SSL configuration repertoire with an ID, a defined keystore, and an optional truststore. PID is com.ibm.ws.ssl.repertoire.

keyStoreRef
A keystore containing key entries for the SSL configuration repertoire. This attribute is required.

string

true

trustStoreRef
A keystore containing trusted certificate entries used by the SSL configuration repertoire for signing verification. This attribute is optional. If unspecified, the same keystore is used for both key and trusted certificate entries.

string

${keyStoreRef}

false

sslDefault

The default configuration repertoire for SSL services. PID is com.ibm.ws.ssl.default.

sslRef
The default SSL configuration repertoire. The default value is defaultSSLSettings.

string

defaultSSLConfig

false

sslOptions

The SSL protocol configuration for a transport. PID is com.ibm.ws.sslchannel.options.

sessionTimeout
Amount of time to wait for a read or write request to complete on a socket. This value is overridden by protocol-specific timeouts. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

1d

false

sslRef
default.sslRef.desc

string

false

tcpOptions

Defines TCP protocol settings. PID is com.ibm.ws.tcpchannel.options.

`inactivityTimeout`

Amount of time to wait for a read or write request to complete on a socket. This value is overridden by protocol-specific timeouts. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

60s

false

`soReuseAddr`

Enables immediate rebind to a port with no active listener.

boolean

false

false

transaction

Configuration properties for the Transaction Manager service. PID is com.ibm.ws.transaction.

`recoverOnStartup`

Specifies whether the server should begin transaction recovery at server startup.

boolean

false

false

`waitForRecovery`

Specifies whether the server should wait for transaction recovery to complete before accepting new transactional work.

boolean

false

false

`acceptHeuristicHazard`

Specifies whether all applications on this server accept the possibility of a heuristic hazard occurring in a two-phase transaction that contains a one-phase resource.

boolean

true

false

`clientInactivityTimeout`

Maximum duration between transactional requests from a remote client. Any period of client inactivity that exceeds this timeout results in the transaction being rolled back in this application server. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

60s

false

heuristicRetryInterval
Amount of time that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

60s

false

heuristicRetryWait
The number of times that the application server retries a completion signal, such as commit or rollback. Retries occur after a transient exception from a resource manager or remote partner.

int

5

false

propogatedOrBMTTranLifetimeTimeout
Upper limit of the transaction timeout for transactions that run in this server. This value should be greater than or equal to the value specified for the total transaction timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

totalTranLifetimeTimeout
Default maximum time allowed for transactions started on this server to complete. Any such transactions that do not complete before this timeout occurs are rolled back. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

12000s

false

transactionLogDirectory
A directory for this server where the transaction service stores log files for recovery.

string

${server.config.dir}/tranlog/

false

transactionLogSize
Specifies the size of transaction log files in Kilobytes.

int

1024

false

`enableLoggingForHeuristicReporting`

Specifies whether the application server logs about-to-commit-one-phase-resource events from transactions that involve both a one-phase commit resource and two-phase commit resources.

boolean

false

false

`timeoutGracePeriodEnabled`

Specifies whether there is a delay between a transaction timeout and the abnormal ending of the servant region that was running the transaction.

boolean

false

false

`lpsHeuristicCompletion`

Specifies the direction that is used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator. Allowed values are: COMMIT, ROLLBACK and MANUAL

string

ROLLBACK

ROLLBACK

COMMIT

MANUAL

false

`defaultMaxShutdownDelay`

Default maximum shutdown delay. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

2s

false

trustAssociation

Controls the operation of the trust association interceptor (TAI) service. PID is com.ibm.ws.security.authentication.tai.

`invokeForUnprotectedURI`

Controls whether the TAI is invoked for an unprotected URI.

boolean

false

true

`failOverToAppAuthType`

Allow an interceptor to fall back to the application authentication mechanism.

boolean

false

true

interceptors
Defines a trust association interceptor.

enabled
Enables or disables the interceptor.

boolean

true

true

className
Fully-qualified package name of the interceptor class.

string

true

invokeBeforeSSO
Invoke an interceptor before single sign-on (SSO).

boolean

true

true

invokeAfterSSO
Invoke an interceptor after single sign-on (SSO).

boolean

false

true

libraryRef
A reference to the ID of the shared library configuration.

Configuration ID of type library (string).

false

library
A reference to the ID of the shared library configuration.

Element of type library.

false

properties
false

true

virtualHost

Virtual host configuration. PID is com.ibm.ws.http.virtualhost.

enabled
Enable this virtual host.

boolean

true

false

webAppSecurity

Configures web container application security. PID is com.ibm.ws.webcontainer.security.internal.WebAppSecurityCollaboratorImpl.

allowFailOverToBasicAuth

Specifies whether to fail over to basic authentication when certificate authentication fails. The equivalent custom property in the full application server profile is com.ibm.wsspi.security.web.failOverToBasicAuth.

boolean

false

false

allowLogoutPageRedirectToAnyHost

Warning, security risk: Setting this property to true may open your systems to potential URL redirect attacks. If set to true, any host can be specified for the logout page redirect. If set to false, and the logout page points to a different host, or one not listed in the logout page redirect domain list, then a generic logout page is displayed. The equivalent custom property in the full application server profile is com.ibm.websphere.security.allowAnyLogoutExitPageHost.

boolean

false

false

displayAuthenticationRealm

Warning, security risk: if this property is set to true, and the user registry's realm name contains sensitive information, it is displayed to the user. For example, if an LDAP configuration is used, the LDAP server hostname and port are displayed. This configuration controls what the HTTP basic authentication login window displays when the realm name is not defined in the application web.xml. If the realm name is defined in the application web.xml file, this property is ignored. If set to true, the realm name displayed will be the user registry realm name for the LTPA authentication mechanism or the Kerberos realm name for the Kerberos authentication mechanism. If set to false, the realm name displayed will be "Default Realm". The equivalent custom property in the full application server profile is com.ibm.websphere.security.displayRealm.

boolean

false

false

httpOnlyCookies

Specifies whether the HTTP only (HttpOnly) cookies option is enabled.

boolean

true

false

logoutOnHttpSessionExpire

Specifies whether users will be logged out after the HTTP session timer expires. If set to false, the user credential will stay active until the Single Sign-On token timeout occurs. The equivalent custom property in the full application server profile is com.ibm.ws.security.web.logoutOnHTTPSessionExpire.

boolean

false

false

logoutPageRedirectDomainNames

A pipe (|) separated list of domain names that are allowed for the logout page redirect (localhost is implied). The equivalent custom property in the full application server profile is com.ibm.websphere.security.logoutExitPageDomainList.

string

false

singleSignonEnabled

Specifies whether single sign-on is enabled.

boolean

true

false

ssoDomainNames

A pipe (|) separated list of domain names that SSO Cookies should be presented. The equivalent custom property in the full application server profile is com.ibm.ws.security.config.SingleSignonConfig

string

false

ssoUseDomainFromURL

Specifies whether to use the domain name from the request URL for the cookie domain.

boolean

false

false

ssoRequiresSSL

Specifies whether a SSO cookie is sent over SSL. The equivalent custom property in the full application server profile is com.ibm.websphere.security.customSSOCookieName

boolean

false

false

preserveFullyQualifiedReferrerUrl

Warning, security risk: Setting this to true may open your systems to potential URL redirect attacks. This property specifies whether the fully qualified referrer URL for form login redirects is preserved. If false, the host for the referrer URL is removed and the redirect is to localhost. The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.fullyQualifiedURL

boolean

false

false

postParamSaveMethod

Specifies where POST parameters are stored upon redirect. Valid values are cookie (POST parameters are stored in a cookie), session (POST parameters are stored in the HTTP Session) and none (POST parameters are not preserved). The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.postParamSaveMethod.

string

Cookie

Cookie

Session

None

false

`postParamCookieSize`

Size of the POST parameter cookie. If the size of the cookie is larger than the browser limit, unexpected behavior may occur. The value of this property must be a positive integer and represents the maximum size of the cookie in bytes. The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.postParamMaxCookieSize.

int

16384

false

`ssoCookieName`

Customizes the SSO cookie name. A custom cookie name allows you to logically separate authentication between SSO domains and to enable customized authentication to a particular environment. Before setting this value, consider that setting a custom cookie name can cause an authentication failure. For example, a connection to a server that has a custom cookie property set sends this custom cookie to the browser. A subsequent connection to a server that uses either the default cookie name or a different cookie name, is not able to authenticate the request via a validation of the in-bound cookie. The equivalent custom property in the full application server profile is com.ibm.websphere.security.customSSOCookieName.

string

LtpaToken2

false

`webAlwaysLogin`

Specifies whether the login() method will throw an exception when an identity has already been authenticated.

boolean

false

false

`useAuthenticationDataForUnprotectedResource`

Specifies whether authentication data can be used when accessing an unprotected resource. The unprotected resource can access validated authenticated data that it previously could not access. This option enables the unprotected resource to call the getRemoteUser, isUserInRole, and getUserPrincipal methods to retrieve an authenticated identity. The equivalent custom property in the full application server profile is com.ibm.wsspi.security.web.webAuthReq=persisting.

boolean

true

false

webContainer

Configuration for the web container. PID is com.ibm.ws.webcontainer.

`listeners`

A comma separated list of listener classes.

string

false

decodeUrlAsUtf8
Decode URL using the an encoding setting of UTF-8.

boolean

true

false

fileServingEnabled
Enable file serving if this setting was not explicitly specified for the application.

boolean

true

false

disallowAllFileServing
Disables all file serving by applications. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.disallowAllFileServing.

boolean

false

false

directoryBrowsingEnabled
Enable directory browsing of an application.

boolean

false

false

serveServletsByClassnameEnabled
Enable servlets to be accessed in a web application using a class name if not explicitly specified.

boolean

false

false

disallowServeServletsByClassName
Disallows the use of serveServletsByClassnameEnabled on the application server level. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.disallowserveservletsbyclassname.

boolean

false

false

doNotServeByClassName
A semi-colon delimited list of classes to be completely disallowed from being served by classname. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.donotservebyclassname.

string

false

trustHostHeaderPort
Set this property to true and the com.ibm.ws.webcontainer.extractHostHeaderPort custom property to true to return the port number from the request host header first.

boolean

false

false

trusted
Enables the application server to use inbound private headers from the web server plug-in.

boolean

true

false

extractHostHeaderPort
The web container will return a port number from the host header, if any, or the URL port on which the client connection was accepted. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.extracthostheaderport.

boolean

false

false

httpsIndicatorHeader
For SSL offloading, set to the name of the HTTP header variable inserted by the SSL accelerator/proxy/load balancer.

string

false

exposeWebInfOnDispatch
If true, a servlet can access files in the WEB-INF directory. If false (default), a servlet cannot access files the WEB-INF directory.

boolean

false

false

decodeUrlPlusSign
Decode the plus sign when it is part of the URL. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.decodeurlplussign.

boolean

false

false

channelWriteType
When set to 'sync', responses will be written synchronously; otherwise, responses will be written asychronously. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.channelwritetype.

string

async

false

suppressHtmlRecursiveErrorOutput
Suppresses the HTML output of a recursive error that cannot be handled by an application's configured error page. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.suppressHtmlRecursiveErrorOutput.

boolean

false

false

`fileWrapperEvents`

Web container will generate SMF and PMI data when serving the static files. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.fileWrapperEvents.

boolean

false

false

`webGroupVhostNotFound`

Replaces message SRVE0017W or SRVE0255E with a user-defined string. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.webgroupvhostnotfound.

string

false

`defaultTraceRequestBehavior`

Restore HTTP TRACE processing. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.DefaultTraceRequestBehavior.

boolean

false

false

`defaultHeadRequestBehavior`

Restore the behavior where the HEAD request is not subject to the security constraint defined for the GET method. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.DefaultHeadRequestBehavior.

boolean

false

false

`tolerateSymbolicLinks`

Enables the web container to support the use of symbolic links. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.TolerateSymbolicLinks.

boolean

false

false

`symbolicLinksCacheSize`

Initial size of the symbolic link cache. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.SymbolicLinksCacheSize.

int

1000

false

`enableErrorExceptionTypeFirst`

Web container is updated to search and use the exception-type before the error-code. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enableErrorExceptionTypeFirst.

boolean

false

false

enableMultiReadOfPostData

Retain post data for multiple read accesses. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enablemultireadofpostdata.

boolean

false

false

copyAttributesKeySet

Web container will return an enumeration of a copy of the list attributes to the servlet to avoid a concurrent access error by the servlet. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.copyattributeskeyset.

boolean

false

false

dispatcherRethrowsEr

Web container will re-throw errors allowing interested resources to process them. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.dispatcherRethrowser.

boolean

true

false

ignoreSessiononStaticFileRequest

Improves performance by preventing the web container from accessing a session for static file requests involving filters. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.IgnoreSessiononStaticFileRequest.

boolean

false

false

invokeFilterInitAtStartup

Web container will call the filter's init() method at application startup. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.invokeFilterInitAtStartup.

boolean

true

false

enableJspMappingOverride

Allow the JSP mapping to be overridden so that the application can serve the JSP contents itself. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enablejspmappingoverride.

boolean

false

false

enableDefaultIsElIgnoredInTag

Correct the default behavior of the EL expression's evaluation in the tag files. The equivalent custom property in the full application server profile is com.ibm.ws.jsp.enabledefaultiselignoredintag.

boolean

false

false

parseUtf8PostData

Web container will detect non URL encoded UTF-8 post data and include it in the parameter values. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.parseutf8postdata.

boolean

false

false

logServletContainerInitializerClassLoadingErrors

Log servlet container class loading errors as warnings rather than logging them only when debug is enabled. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.logservletcontainerinitializerclassloadingerrors.

boolean

false

false

allowIncludeSendError

Allow RequestDispatch to send errors on Include methods. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.allowincludesenderror.

boolean

false

false

skipMetaInfResourcesProcessing

Do not search the meta-inf directory for application resources. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.skipmetainfresourcesprocessing.

boolean

false

false

metaInfResourcesCacheSize

Initial size (number of entries) of the meta-inf resource cache. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.metainfresourcescachesize.name.

int

20

false

xPoweredBy

Alternative string for the X-Powered-By header setting. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.xpoweredby.

string

false

disableXPoweredBy

Disable setting of the X-Powered-By header. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.disablexpoweredby.

boolean

false

false

deferServletLoad

Defer servlet loading and initialization until the first request.

boolean

true

false

asyncMaxSizeTaskPool

Maximum size of tasks in the Async task pool before automatically purging canceled tasks. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asyncmaxsizetaskpool.

int

5000

false

asyncPurgeInterval

Time interval to wait between each required purge of the cancelled task pool. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asyncpurgeinterval.

int

30000

false

asyncTimeoutDefault

Async servlet timeout value used when a timeout value has not been explcitly specified. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asynctimeoutdefault.

int

30000

false

asyncTimerThreads

Maximum number of threads to use for async servlet timeout processing. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asynctimerthreads.

int

2

false

wlmClassification

zos.wlm.classification.config.description. PID is com.ibm.ws.zos.wlm.classification.

classificationName

optional classification name

string

null

false

zosWorkloadManager

WLM Native Enclave Manager Configuration. PID is com.ibm.ws.zos.wlm.WLMConfigManager.

collectionName

A collection name to be used for classifying work requests. The name is supplied to native services IWMCLSFY or __server_classify. Only the first 18 characters will be used.

string

${wlp.server.name}

false

Liberty profile: Feature management

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server.

The configuration file server.xml declares which features you want to load. The set of features is enclosed within the <featureManager> element, and each feature within the <feature> sub-element. For example:

```
<server>
    <featureManager>
        <feature>servlet-3.0</feature>
        <feature>localConnector-1.0</feature>
    </featureManager>
</server>
```

You can specify any feature in the server configuration. Some features include other features within them. The same feature can be included in one or more other features. At runtime the feature manager computes the combined list of content that is required to support the requested set of features.

For information about the main available features, see “Liberty profile: Server features.” For information about the restrictions that apply to each feature, see Liberty profile: Runtime environment known restrictions.

Dynamic changes to feature configuration

When you change the feature configuration, the feature manager recalculates the list of required bundles, stops and uninstalls those no longer needed, and installs and starts any additions. All features are therefore designed to be able to cope with other features being dynamically added or removed.

Liberty profile: Server features

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server.

The following list contains information about the main available features. Including a feature in the configuration might cause one or more additional features to be loaded automatically. For example, if you include the wab-1.0 feature, the servlet-3.0 and blueprint-1.0 features are loaded automatically. For each feature it includes a brief description, and an example of how the feature is declared within the <featureManager> element inside the server.xml file. For example:

```
<server>
    <featureManager>
        <feature>servlet-3.0</feature>
        <feature>localConnector-1.0</feature>
    </featureManager>
</server>
```

For a full list of available features, see the `.mf` files under the `lib/features` directory of the server root installation location. The feature information is given in the `Subsystem-Content` element in the `.mf` file. Each `.mf` file represents a feature in the Liberty profile. The file name matches the feature name. For example, the `servlet-3.0` feature is defined in a file called `servlet-3.0.mf`.

Bean validation

`<feature>beanvalidation-1.0</feature>`

The `beanvalidation-1.0` feature provides validations for JavaBeans at each layer of an application.

The validation can be applied to all layers of JavaBeans in an application using either annotations or a `Validation.xml` deployment descriptor.

See also beanvalidation-1.0 feature restrictions

Blueprint

`<feature>blueprint-1.0</feature>`

The `blueprint-1.0` feature enables support for deploying OSGi applications that make use of the OSGi blueprint container specification.

The OSGi Applications support in WebSphere Application Server helps you develop and deploy modular applications that use both Java EE and OSGi technologies. See OSGi Applications.

Java API for RESTful Web Services

`<feature>jaxrs-1.1</feature>`

The `jaxrs-1.1` feature provides support for Java API for Restful Web Services of the Liberty profile.

See also jaxrs-1.1 feature restriction.

Java Database Connectivity (JDBC)

`<feature>jdbc-4.0</feature>`

The `jdbc-4.0` feature provides support for applications that access a database. You can take an existing application that uses Java Database Connectivity (JDBC) and a data source, and deploy the application to a server.

See also Deploying an existing JDBC application to the Liberty profile.

Java Naming and Directory Interface

`<feature>jndi-1.0</feature>`

The `jndi-1.0` feature provides support for a single JNDI entry definition in the server configuration of the Liberty profile.

Java Persistence API (JPA)

`<feature>jpa-2.0</feature>`

The `jpa-2.0` feature provides support for applications that use application-managed and container-managed JPA written to the JPA 2.0 specification.

Support is built on top of Apache OpenJPA with extensions to support the container-managed programming model.

See also Deploying a JPA application to the Liberty profile and jpa-2.0 feature restrictions.

Java Server Faces (JSF)

```
<feature>jsf-2.0</feature>
```

The `jsf-2.0` feature provides support for web applications that use the JSF framework. This framework simplifies the construction of user interfaces.

If you include the `jsf-2.0` feature you are also including the `jsp-2.2` feature, because the JSF framework is an extension of the JSP framework.

See also jsf-2.0 feature restrictions.

Java Server Pages (JSP)

```
<feature>jsp-2.2</feature>
```

The `jsp-2.2` feature provides support for JSPs written to the JSP 2.2 specification.

If you include the `jsp-2.2` feature you are also including the `servlet-3.0` feature.

See also jsp-2.2 feature restrictions.

JavaScript Object Notation (JSON4) Library

```
<feature>json-1.0</feature>
```

The `json-1.0` feature provides access to the JSON4J library, which provides a set of JSON handling classes for use within Java environments. The JSON4J library provides a simple Java model for constructing and manipulating data to be rendered as JSON data.

See also Using JSON content in JAX-RS application requests and responses and JSON4J Libraries API.

Local JMX Connector

```
<feature>localConnector-1.0</feature>
```

The `localConnector-1.0` feature provides a local JMX connector that is built into the JVM and can only be used on the same host machine by someone running under the same user ID and the same JDK. It enables local access by JMX clients such as jConsole, or other JMX client that use the Attach API.

See Accessing JMX connectors on the Liberty profile.

Monitoring

```
<feature>monitor-1.0</feature>
```

The `monitor-1.0` feature provides Performance Monitoring Infrastructure (PMI) support on the Liberty profile.

See Monitoring the Liberty profile.

OSGi JPA

```
<feature>osgi.jpa-1.0</feature>
```

The `osgi.jpa-1.0` feature provides JPA support for OSGi applications on the Liberty profile.

Remote JMX connector

```
<feature>restConnector-1.0</feature>
```

The `restConnector-1.0` feature provides a secure JMX connector that can be used locally or remotely using any JDK. It enables remote access by JMX clients via a REST based connector and requires SSL and basic user security configuration.

See Accessing JMX connectors on the Liberty profile.

Secure Sockets Layer (SSL)

```
<feature>ssl-1.0</feature>
```

The `ssl-1.0` feature provides support for Secure Sockets Layer (SSL) connections. To use the secure HTTPS listener, you must enable this feature.

The Liberty profile provides a dummy keystore and a dummy truststore, which are the same as those provided by previous versions of WebSphere Application Server.

The secure HTTPS listener is not started unless the `ssl-1.0` feature is enabled. If the feature is unavailable, the HTTPS listener is stopped.

To specify the SSL certificates to use, you add a pointer into the `server.xml` file. See Securing communications with the Liberty profile.

To change the HTTPS port, set the `<httpsPort>` attribute of the `<httpEndpoint>` element in the `server.xml` file. See Specifying Liberty profile bootstrap properties.

Security

```
<feature>appSecurity-1.0</feature>
```

The `appSecurity-1.0` feature provides support for securing the server runtime environment and applications. The following aspects are supported:

- Basic user registry
- Lightweight Directory Access Protocol (LDAP) user registry
- Basic authorization
- Web application security
  - Basic authentication login
  - Form-login Form-logout
  - Programmatic APIs: getRemoteUser; getUserPrincipal; isUserInRole; authenticate; logout; login

When you add the `appSecurity-1.0` feature to your server, you also need to configure either the basic user registry or the LDAP user registry.

See Securing the Liberty profile and its applications.

Server status

```
<feature>serverStatus-1.0</feature>
```

The `serverStatus-1.0` feature enables Liberty profile servers to automatically publish their status to WebSphere Application Server Deployment Managers and Job Managers that are aware of the server as a resource in their Job configuration. The known states are **`Started`** and **`Stopped`**.

See Submitting jobs to manage Liberty profile servers and Installing Liberty profile server resources using the job manager.

Servlet

```
<feature>servlet-3.0</feature>
```

The `servlet-3.0` feature provides support for HTTP Servlets written to the Java Servlet 3.0 specification. Support is achieved through use of the WebSphere web container.

See also Securing the Liberty profile and its applications.

Session Persistence

```
<feature>sessionDatabase-1.0</feature>
```

The `sessionDatabase-1.0` feature provides session affinity and failover support on the Liberty profile.

See Configuring session persistence for the Liberty profile.

Web application bundle (WAB)

```
<feature>wab-1.0</feature>
```

The `wab-1.0` feature provides support for WABs that are inside enterprise bundles. This feature supports the following resources packaged inside a WAB: static web content and JSPs; HTTP servlets written to the Servlet 3.0 specification; blueprint applications.

If you include the `wab-1.0` feature you are also including the `servlet-3.0` and `blueprint-1.0` features.

Liberty profile: Security

This topic provides an overview of security workflow in the Liberty profile.

Liberty profile security provides protection for both web resources as per the Servlet 3.0 specification as well as protecting the JMX connections when using the REST connector.

The following diagram shows a typical security process involved when accessing a protected web resource. In order for this to work, one must configure the appropriate security features and the configuration required for the authentication and authorization to happen.

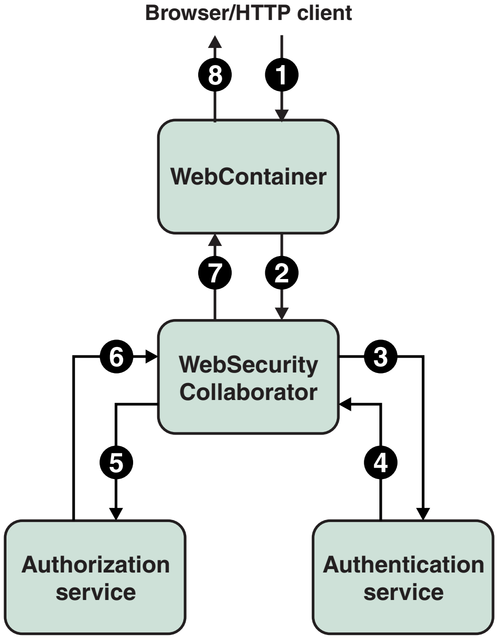


Figure 19. Typical security flow for web resources

1. A HTTP client requests a web resource in the WebContainer.
2. The WebContainer delegates the security check to the WebSecurity Collaborator.
3. The WebSecurity Collaborator prompts the user to enter credentials (if absent), and uses the Authentication service to authenticate the user.
4. The Authentication service authenticates, creates, and returns the subject if authenticated successfully. Otherwise, the Authentication Service reports an exception for the authentication failure.
5. The WebSecurity Collaborator uses the Authorization service to perform a user authorization check.
6. The Authorization service returns the authorization result to the WebSecurity Collaborator.
7. The WebSecurity Collaborator returns the result of the security check (whether the user is in an authorized role).
8. The WebContainer serves or rejects the requested resource.

The following sections describe all the primary security components in the Liberty profile:

- "Quick start" on page 322
- "Authentication" on page 322

- “Authorization”
- “Secure Socket Layer (SSL)”
- “Single Sign-On (SSO)”
- “Web security related properties”
- “Security Public APIs” on page 323
- “Management security” on page 323
-
- “JCA Security” on page 323
- “Configuration examples and samples” on page 323
- “Security compatibility and differences” on page 323
- “Troubleshooting” on page 323

Quick start

The `quickStartSecurity` element helps you configure a single user security environment in the Liberty profile. See “Liberty profile: Quick overview of security” on page 323 for how the security workflow is when using the `quickStartSecurity` element, and Getting started with security in the Liberty profile for a sample task.

Authentication

The process of confirming the identity of a user is known as authentication. The most common form of authentication is user name and password, such as through either basic authentication or form login for web applications. Once a user is authenticated, the source of a request is represented as a `Subject` object at the run time. This process involves performing access control checks when a user accesses a resource, based on the authorization rules configured for the resource. See “Liberty profile: Authentication” on page 324 for more concepts and Authenticating users in the Liberty profile for detailed tasks.

Authorization

The process of determining whether or not to grant a user access to resources within the system is known as authorization. The Java EE model uses subjects, resources and roles to determine what can and can not be allowed. This process involves checking the user credentials (such as the user ID and password, certificates and tokens), and creating a subject based on the authenticated user. See “Liberty profile: Authorization” on page 333 for more concepts and Authorizing access to resources in the Liberty profile for detailed tasks.

Secure Socket Layer (SSL)

SSL provides transport level security. See Enabling SSL communication for the Liberty profile for detailed tasks.

Single Sign-On (SSO)

SSO enables access to applications without being prompted to login multiple times. See Concept of SSO for more details and Customizing SSO configuration using LTPA cookies for the Liberty profile for the detailed task.

Web security related properties

There are many configuration properties that you can configure as part of web security such as SSO and client certificate authentication for your applications. See “Liberty profile: Configuration elements in the server.xml file” on page 195 for available attributes and see Configuring web security related properties for the Liberty profile for some examples.

Security Public APIs

The Liberty profile contains public APIs that you can use to implement security functions. The security public APIs in the Liberty profile are a subset of the full profile security public APIs. The main classes are `WSSecurityHelper`, `WSSubject`, and `RegistryHelper`. These classes contain a subset of the methods that are available in the full profile versions. There is also a new class `WebSecurityHelper`. See “Liberty profile: Security public APIs” on page 335.

The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

See Developing extensions to the Liberty profile security infrastructure for some of examples.

Management security

Management security means you can manage the Liberty profile using a remote JMX client. To secure remote connections using the REST connector, see Accessing JMX connectors on the Liberty profile. Also you can develop your own JMX client application as in Developing a JMX Java client for the Liberty profile.

JCA Security

The Java Connector Architecture (JCA) is to provide the security support for database connectivity. See Configuring JCA security for the Liberty profile

Configuration examples and samples

There are several security configuration examples under the `${wlp.install.dir}/templates/config` directory to help you understand and develop common security configurations. You can refer to those examples when configuring security for your application on the Liberty profile.

Security compatibility and differences

You can learn about the main differences in the security capability between the full profile and the Liberty profile. See “Configuration differences between the full profile and Liberty profile: security” on page 339.

Troubleshooting

The troubleshooting information can be used to solve security-related problems when using the Liberty profile. See Troubleshooting security and Troubleshooting LDAP.

Liberty profile: Quick overview of security

This topic describes some of common security terms, along with an example which helps you understand the basic workflow of security in the Liberty profile.

Security key terms

Authorization

The process of determining whether or not to grant a user access to resources within the system is known as authorization. The Java EE model uses subjects, resources and roles to determine what should and should not be allowed.

Authentication
The process of confirming the identity of a user is known as authentication. The most common form of authentication is user name and password, such as through either basic authentication or form login for web applications. Once a user is authenticated, the source of a request is represented as a `Subject` object at the run time.

Resource
Also known as an object, resources are things within the system. A resource can be any non-active entity, such as a web application.

Role A role is a logical collection of privileges that can be assigned to a user or group. Some roles are predefined by the system (such as the Administrator role). Others are defined by the application developer. In Java EE, subjects are usually granted or denied access to resources based on the roles they do (or do not) possess.

Subject
A subject is both a general term, as well as a Java object `javax.security.auth.Subject`. Generally, the term subject means active entities within the system, such as users on the system, and even the system process itself.

Security workflow

The following example demonstrates how the security works when a user requests access to a resource. For example, a user **`Bob`** wants to access a servlet `myWebApp`. See the code samples in Getting started with security in the Liberty profile.

In order to do this, the following conditions must be true:

1. **`Bob`** must be able to log into the system because the servlet is protected.
2. **`Bob`** must be in the `testing` role because the servlet is restricted using an `auth-constraint` element in the deployment descriptor.

If **`Bob`** cannot log into the system, or **`Bob`** is not in the `testing` role, then the access to the servlet `myWebApp` is denied.

Another user **`Alice`** can log into the system because **`Alice`** is a valid user. But **`Alice`** is not in the `testing` role. An HTTP 403 error (Access Denied/Forbidden) shows up when **`Alice`** logs in.

Liberty profile: Authentication

This topic provides an overview of authentication in the Liberty profile.

When a protected web resource is accessed, the user must provide credential data, such as user ID and password, to access the resource. The authentication process involves collecting this user credential information (based on how the web application was configured to collect this data) and validating it against the configured registry. When it is verified, a JAAS subject is created for that user. The subject contains additional information about the user, such as the groups that the user belongs to, and the tokens created for the user. The information in this subject is then used during the authorization process to determine whether the user can access the resource.

The following diagram illustrates a typical authentication process flow for a web resource.

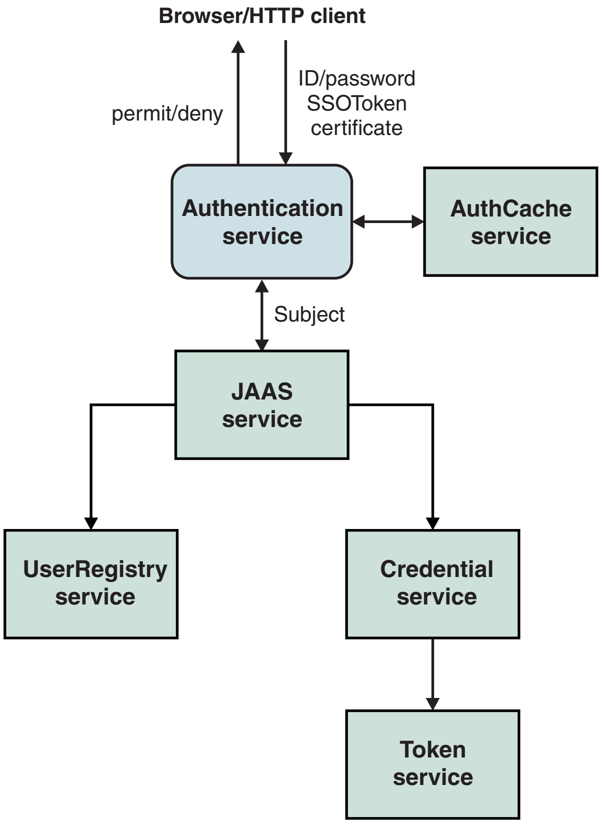


Figure 20. Overview of authentication process

The authentication process involves gathering credential data from the user, checking the cache to see whether the subject exists for that user and in its absence calling the JAAS service to perform the authentication to create a subject. The JAAS service calls a set of login modules to handle the authentication. One or more of the login modules handles the creation of the subject depending on the credential data. The login module then calls the registry that is configured to validate the credential information. If the validation is successful, the authentication process collects and creates relevant information for that user, including the groups that the user belongs to and the Single Sign On (SSO) token used for SSO capability, and stores them in the subject as relevant credentials. You can also customize the information saved in the subject by plugging in custom login modules during this process.

When the authentication is successful, the SSO token that is created during the process is sent back to the browser in a cookie. The default name of the cookie is `ltpaToken2` and is configurable. On subsequent calls, the token information is used to authenticate the user. If this authentication fails, the authentication service tries to use other authentication data, such as the user ID and password, if they still exist in the request.

The following sections describe each of these concepts in more details:

- “User registries” on page 326
- “Authentication Cache” on page 326
- “JAAS configuration” on page 326
- “JAAS login modules” on page 326
- “Callback handler” on page 327
- “Credentials/Tokens” on page 328
- “LTPA” on page 328

- “Single Sign-On (SSO)” on page 329
- “Pluggable authentication” on page 329
- “Identity assertion” on page 329
- “RunAs() authentication” on page 330
- “Proxy login module” on page 331
- “Certificate login” on page 331
- “Hashtable login module” on page 331

User registries

When validating the authentication data of a user, the login modules call the user registry that is configured to validate the user information. The Liberty profile supports both a simple configuration-based user registry and a more robust LDAP-based repository. For more information, see Configuring a user registry for the Liberty profile.

Authentication Cache

Because the creation of a subject is relatively expensive, the Liberty profile provides an authentication cache to store a subject after an authentication of a user is successful. The default expiration time for the cache is 10 minutes. If the user does not log back in within 10 minutes, the subject is removed and the process of authentication repeats to create a subject for that user. You can configure the cache timeout period, the cache size, and also disable or enable caching. For more information, see Configuring the authentication cache on the Liberty profile.

JAAS configuration

A JAAS configuration defines a set of login modules that are used to create the subject. The Liberty profile supports the following JAAS configurations:

system.WEB_INBOUND
This JAAS configuration is used when accessing web resources such as servlets and JSPs.

WSLogin
This JAAS configuration is used by applications when using the programmatic login.

system.DEFAULT
This JAAS configuration is used for login when no JAAS configuration is specified.

Note:

- The system.WEB_INBOUND and system.DEFAULT configurations have these default login modules in this order: **hashtable**, **userNameAndPassword**, **certificate**, and **token**.
- The WSLogin configuration has the proxy login module as the default login module, and the proxy delegates all the operations to the real login module in system.DEFAULT.

No explicit configuration is needed unless you want to customize with custom login modules. Depending on the requirement, you can customize specific login configurations. For example, if you want all the web resource logins to be customized, then custom login modules must be added only to the `system.WEB_INBOUND` configuration. See Configuring a JAAS custom login module for the Liberty profile.

JAAS login modules

The JAAS configuration uses a set of login modules to create the subject. The Liberty profile provides a set of login modules in each of the login configurations. Depending on the authentication data, a particular login module handles the creation of the subject. The authentication data is passed to the login modules using the callback handler as specified in the JAAS specification. For example, if the user ID and password callback handler is being used for authentication, the **userNameAndPassword** login module handles the authentication. If a `SingleSignonToken` is presented as the authentication data, only the **token** login module handles the authentication.

The following default login modules are supported in the Liberty profile:

userNameAndPassword
Handles the authentication when user name and password are used as the authentication data.

certificate
Handles the authentication when an X509 certificate is used as the authentication data as in the case of mutual SSL.

token Handles the authentication when an SSO token is presented as the authentication data. During the authentication process, an SSO token is created and sent back to the http client (browser) in a cookie. On subsequent requests, this cookie is sent back by the browser and the server extracts the token from the cookie to authenticate the user when single sign-on is enabled.

hashtable
Is used when the authenticated data is sent through a predefined hashtable. For more information about the hashtable login, see “Hashtable login module” on page 331. This login module is also used by the security run time when authentication is performed using identity only. For example, in the case of `runAs`.

proxy Is the default login module for WSLogin. See “Proxy login module” on page 331.

The login modules are called in the order that they are configured. The default order is **hashtable**, **userNameAndPassword**, **certificate**, **token**. If you must customize the login process using custom login modules you can provide them and configure them in the order you need. Typically, a custom login module would be placed first in the list of login modules so that it is called first. When a custom login module is used, you must specify all the login module information in the configuration along with the custom login module in the order required.

When a login module determines that it can handle the authentication, it first makes sure that the authentication data that is passed in is valid. For example, for user name and password authentication, the configured user registry is called to verify the authentication information. For token authentication, the token must be decrypted and valid for the verification to succeed.

When the authentication data is validated, the login modules create credentials with additional data for the user including the groups and the SSO token. A custom login module can add additional data to the subject by creating its own credentials. For the Liberty profile authorization to work, the subject must contain the `WSCredential`, `WSPrincipal`, and `SingleSignonToken` credentials. The `WSCredential` credential contains the groups information, along with additional information needed by the security runtime environment.

Callback handler

The Liberty profile supports various callback handlers for providing data to the login modules during the JAAS authentication process. A custom login module can use the callback handler information to perform its own authentication. For example, if the callback handler needs to access some information in an `HttpServletRequest` object, it can do so using that specific callback handler.

The following callback handlers and factories for programmatic JAAS login are supported in the Liberty profile:

1. `com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl`
2. `com.ibm.wsspi.security.auth.callback.WSCallbackHandlerFactory`

The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

See Developing JAAS custom login modules for a system login configuration.

Credentials/Tokens

As mentioned in the loginModule section, credentials are created as part of the subject creation process. The Liberty profile creates the `WSCredential`, `SingleSignonToken`, and `WSPrincipal` credentials. The `SingleSignonToken` credential contains the token that is sent back to the browser in a cookie for SSO to work. This token contains both the user information and an expiration time. It is signed and encrypted using the Lightweight Third Party Authentication (LTPA) keys that are generated during the first server startup. The default expiration time is 2 hours and is an absolute time (not based on user activity). After the 2 hours, the token expires and the user must log in again to access the resource.

LTPA

LTPA is intended for distributed, multiple application server environments. In the Liberty profile, LTPA supports SSO and security in a distributed environment through cryptography. This support enables LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

Application servers can securely communicate using the LTPA protocol. It also provides the SSO feature, whereby a user is required to authenticate only when connecting to a domain name system (DNS) domain, and can then access resources in other Liberty profile servers in the same domain without getting prompted. The realm names on each system in the DNS domain are case-sensitive and must match identically.

The LTPA protocol uses cryptographic keys to encrypt and decrypt user data that passes between the servers. These keys must be shared between the different servers for the resources in one server to access resources in other servers, assuming that all the servers involved use the same user registry. LTPA requires that the configured user registry must be a centrally shared repository so that users and groups are the same, regardless of the server.

When using LTPA, a token is created that contains both the user information and an expiration time, and is signed by the keys. The LTPA token is time sensitive. All participating servers must have their time and date synchronized. If not, LTPA tokens are prematurely expired and cause authentication or validation failures. Coordinated Universal Time (UTC) is used by default, and all other servers must have the same UTC time. Consult your operating system documentation for information about how to ensure the same UTC time among servers.

This token passes to other servers through cookies for web resources when SSO is enabled.

If the receiving servers use the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure that it is not expired and that the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

Each server must have valid credentials. When the credentials expire, the server is required to communicate to the user registry to authenticate. Extending the time the LTPA token remains cached does present a slightly increased security risk to be considered when defining your security policies.

If key sharing is required between different Liberty profile servers, copy the keys from one server to another. For security purposes, the keys are encrypted with a randomly-generated key and a user-defined password is used to protect the keys. This same password is needed when importing the keys into another server. The password is only used to protect the keys and is not used to generate the keys.

When security is enabled, LTPA is configured by default during the Liberty profile server start time. For more information about the LTPA support, see Configuring LTPA on the Liberty profile.

Single Sign-On (SSO)

SSO enables user to log in one place (one server for example) and access applications on other servers without getting prompted again. To make SSO work, the LTPA keys must be exchanged across different Liberty profile servers, the user registries must be the same and the token must not have expired. To exchange the LTPA keys, you can copy the `ltpa.keys` file from one server to another and restart the server to use the new LTPA keys. The registries used by all the servers participating in the SSO domain must be the same.

When a user is authenticated in one Liberty profile server, the SSO token created for the user during the authentication process is put in the cookie that is sent to the HTTP client, for example, a browser. If there is another request from that client to another set of applications on a different server but in the same DNS domain that was configured as part of the SSO configuration in the first server the cookie is sent along with the request. The receiving server tries to authenticate the user using the token in the cookie and if both conditions are met, it validates the token and create a subject based for the user in this server without prompting the user to log in again. If for some reason the token cannot be validated (for example, it cannot decrypt or verify the token because of LTPA key mismatch), the user is prompted to enter the credential information again.

Any application that is configured to use `Form-login` requires SSO to be configured for that server. When the user is authenticated for a `form-login`, the token is sent back to the browser which is then used for authorizing the user when accessing the resource.

See Customizing SSO configuration using LTPA cookies for the Liberty profile.

Pluggable authentication

There are various ways to customize the authentication process.

1. Providing a custom login module. See Configuring a JAAS custom login module for the Liberty profile.
2. Implement Trust Association Interceptor to handle all web resource-based authentication. See Developing a custom TAI for the Liberty profile.

Identity assertion

There are various ways to perform identity assertion in the Liberty profile

1. Using the hashtable login. See Developing JAAS custom login modules for a system login configuration.
2. Using the `IdentityAssertionLoginModule`. You can allow an application or system provider to perform an identity assertion with trust validation. To use the `IdentityAssertionLoginModule`, you can use the JAAS login framework, where trust validation is accomplished in one custom login module and credential creation is accomplished in the `IdentityAssertionLoginModule`. The two login modules allow you to create a JAAS login configuration that can be used to perform an identity assertion.

   Two custom login modules are required:

   User implemented login module (trust validation)

   The user implemented login module performs whatever trust verification the user requires. When trust is verified, the trust verification status and the login identity must be put into a map in the share state of the login module so that the credential creation login module can use the information. This map must be stored in the property:

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`
(which consists of)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted`
(which is set to ***true*** if trusted and ***false*** if not trusted)

`com.ibm.wsspi.security.common.auth.module.IdenityAssertionLoginModule.principal`
(which contains the principal of the identity)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates`
(which contains the certificate of the identity)

Identity assertion login module (credential creation)

The `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule` performs the credential creation. This module relies on the trust state information being in the shared state of thelogin context.

The identity assertion login module looks for the trust information in the shared state property,
`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`

, which contains the trust status and the identity to log in and must include:

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted`
(which when ***true*** indicates trusted and ***false*** when not trusted)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal`
(which contains the principal of the identity to log in, if using a principal)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates`
(which contains a array of a certificate chain that contains the identity to log in, if using a certificate)

A `WSLoginFailedException` is returned if the state, trust, or identity information is missing. The login module then performs a login of the identity, and the subject contains the new identity

See Customizing an application login to perform an identity assertion using JAAS.

RunAs() authentication

After calling a servlet and when you have successfully authenticated, the servlet can then make subsequent calls, for example to other servlet. These subsequent calls are normally made under the same security identity that you used to originally log in to the servlet. This identity is known as the caller identity. Alternatively, you can choose to delegate to a different identity by using the RunAs specification so that any subsequent calls the servlet makes runs under this other identity. To summarize, you have two options for propagating the security identity:

- Propagate the caller identity, which is the default behavior.
- Delegate to the runAs identity that you can specify using the runAs specification.

After the server authenticates the original user, the server then authenticates the runAs user. If this authentication fails, then the server falls back to propagating the caller identity.

In order to use the RunAs specification, you must update the deployment descriptor of your application to include the `run-as` element or `@RunAs` annotation. Set this element to the security role that you want to delegate to.

See Configuring RunAs authentication in the Liberty profile.

Proxy login module

The proxy login module class loads the application server login module and delegates all the operations to the real login module implementation. The real login module implementation is specified as the delegate option in the option configuration. The proxy login module is needed because the application class loaders do not have visibility of the application server product shared library class loaders. With an application programmatic login using the `Login()` method of the `LoginContext` class with JAAS login context entry WSLogin, the proxy login module delegates all the work to the JAAS login context entry system.DEFAULT.

Certificate login

The certificate login feature allows you to authenticate web requests such as servlets using client side X509 certificates instead of supplying a user ID and password.

Certificate authentication works by associating a user in the user registry with the distinguished name in the client certificate of a web request. Trust is established by having the client certificate be trusted by the server, for example, the signer of the client certificate must be in the truststore of the server. This mechanism eliminates the need for users to supply a password to establish trust.

See Securing communications with the Liberty profile.

Hashtable login module

Look up the required attributes from the user registry, put the attributes in a hashtable, then add the hashtable to the shared state. If the identity is switched in this login module, you must add the hashtable to the shared state. If the identity is not switched, but the value of the `requiresLogin` code is true, you can create the hashtable of attributes. You do not have to create a hashtable in this situation, because the Liberty profile handles the login for you. However, you might consider creating a hashtable to gather attributes in special cases. For example, if you are using your own special user registry, then creating a `UserRegistry` implementation, using a hashtable, and letting the server gather the user attributes for you might be a simple solution.

The following rules define in more detail how a hashtable login is completed. You must use a `java.util.Hashtable` object in either the Subject (public or private credential set) or the shared-state HashMap. The `com.ibm.wsspi.security.token.AttributeNameConstants` class defines the keys that contain the user information. If the Hashtable object is put into the shared state of the login context using a custom login module that is listed before the hashtable login module, the value of the `java.util.Hashtable` object is searched using the following key within the shared-state hashMap:

Property
com.ibm.wsspi.security.cred.propertiesObject

Reference to the property
AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY

Explanation
This key searches for the Hashtable object that contains the required properties in the shared state of the login context.

Expected result
A java.util.Hashtable object.

If a `java.util.Hashtable` object is found either inside the Subject or within the shared state area, verify that the following properties are present in the hashtable:

- `com.ibm.wsspi.security.cred.uniqueId`

  Reference to the property
  AttributeNameConstants.WSCREDENTIAL_UNIQUEID

Returns
java.util.String

Explanation
The value of the property must be a unique representation of the user. For the Liberty profile default implementation, this property represents the information that is stored in the application authorization table. The information is in the application deployment descriptor after it is deployed and user-to-role mapping is performed. See the expected format examples if the user to role mapping is performed using a lookup to a Liberty profile user registry implementation.

Expected format examples

Table 24. Format examples for uniqueId. This table gives expected format examples.

User Registry	Format (UniqueUserId)
LDAP	ldapRegistryRealm/cn=kevin,o=mycompany,c=use
Basic	basicRegistryRealm/kelvin

The `com.ibm.wsspi.security.cred.uniqueId` property is required.

- `com.ibm.wsspi.security.cred.securityName`

Reference to the property
AttributeNameConstants. WSCREDENTIAL_ SECURITYNAME

Returns
java.util.String

Explanation
This property searches for the **`securityName`** of the authentication user. This name is commonly called the display name or short name. The Liberty profile uses the **`securityName`** attribute for the `getRemoteUser`, `getUserPrincipal`, and `getCallerPrincipal` application programming interfaces (APIs). To ensure compatibility with the default implementation for the **`securityName`** value, call the `public String getUserSecurityName(String uniqueUserId) UserRegistry` method.

Expected format examples

Table 25. Format examples for securityName. This table gives expected format examples.

User Registry	Format (securityName)
LDAP	kevin
Basic	kevin

The `com.ibm.wsspi.security.cred.securityname` property is required.

- `com.ibm.wsspi.security.cred.group`

Reference to the property
AttributeNameConstants. WSCREDENTIAL_GROUP

Returns
java.util.ArrayList

Explanation
This key searches for the array list of groups to which the user belongs. The groups are specified in the **`realm_name/user_name`** format. The format of these groups is important as the groups are used by the Liberty profile authorization engine for group-to-role mappings in the deployment descriptor. The format that is provided must match the format expected by the Liberty profile default implementation. When you use a third-party authorization provider, you must use the format that is expected by the third-party provider. To ensure compatibility with the default implementation for the unique group IDs value, call the `public List getUniqueGroupIds(String uniqueUserId) UserRegistry` method.

Expected format examples

Table 26. Format examples for group. This table gives some format examples when configuring inbound identity mapping.

User Registry	Format (group)
LDAP	ldapRegistryRealm/cn=group1,o=Groups,c=US
Basic	basicRegistryRealm/group1

The `com.ibm.wsspi.security.cred.group` property is required. A user is not required to be associated groups.

- `com.ibm.wsspi.security.cred.cacheKey`

Reference to the property
AttributeNameConstants. WSCREDENTIAL_CACHE_KEY

Returns
java.lang.Object

Explanation
This key property can specify an object that represents the unique properties of the login, including the user-specific information and the user dynamic attributes that might affect uniqueness. For example, when the user logs in from location A, which might affect their access control, the cache key needs to include location A so that the Subject that is received is the correct Subject for the current location.

This `com.ibm.wsspi.security.cred.cacheKey` property is not required. When this property is not specified, the cache lookup is the value that is specified for WSCREDENTIAL_UNIQUEID. When this information is found in the `java.util.Hashtable` object, the Liberty profile creates a Subject similar to the Subject that goes through the normal login process at least for LTPA. The new Subject contains a `WSCredential` object and a `WSPrincipal` object that is fully populated with the information found in the Hashtable object.

Liberty profile: Authorization

This topic provides an overview of authorization in the Liberty profile

Authorization is the function of specifying access rights to resources. It usually follows authentication, which is the function of confirming an identity. Whereas authentication answers the question "Are you who you say you are", authorization answers the question "Do you have permission to do what you are trying to do?"

The following sections describe each of these concepts in more detail:

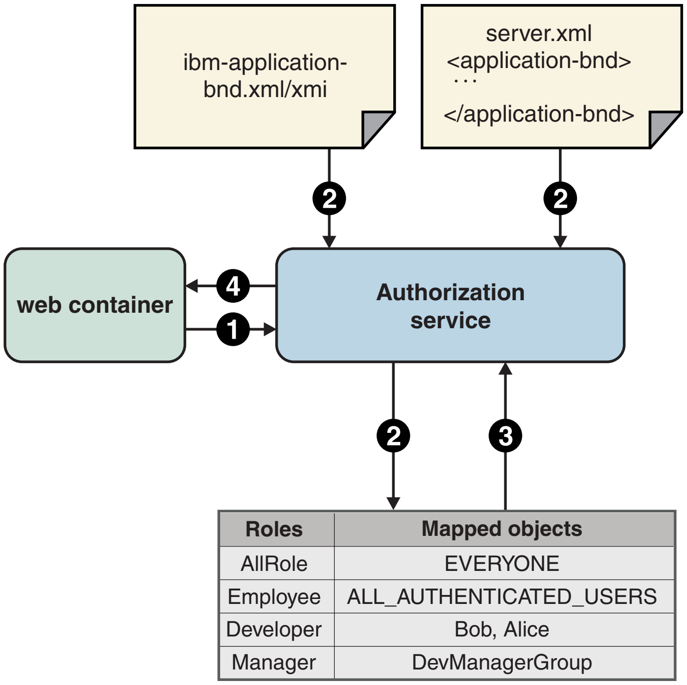
- “Authorization for administrative functions”
- “Authorization for applications” on page 334
- “Special subjects” on page 335
- “Access Ids and authorization” on page 335

Authorization for administrative functions

When an entity attempts to access a resource, the authorization service determines whether that entity does indeed have the required rights to access the resource. This concept holds true whether an entity is accessing an application or performing administrative functions. The main difference between authorizing access to an application and access to an administrative function lies in how the users are mapped to roles. For authorization of applications, you can use the `application-bnd` element in the `server.xml` or the `ibm-application-bnd.xml/xmi` file to map the users to roles. For authorization of administrative functions, use the `administratorRole` element in the `server.xml` file to map the users to the administrator role. For more information about administrative security, See Accessing JMX connectors on the Liberty profile

Authorization for applications

The following diagram describes how authorization works for applications:



Roles	Mapped objects
AllRole	EVERYONE
Employee	ALL_AUTHENTICATED_USERS
Developer	Bob, Alice
Manager	DevManagerGroup

Figure 21. Overview of authorization process

1. Authorization is performed when an entity attempts to access a resource in an application served by the Liberty profile. The web container calls the authorization service to determine whether a user has permission to access a certain resource, given a set of one or more required roles. The required roles are determined by the `auth-constraint` elements in the deployment descriptor and `@ServletSecurity` annotations.
2. The authorization service determines what objects the required role is mapped to. This step is accomplished by processing the mappings defined in the `ibm-application-bnd.xmi` or `ibm-application-bnd.xml` file and the `application-bnd` element of the `server.xml` file. The mappings from these two sources are merged. If the same role is present in both sources, only the role mapping in the `server.xml` file is used. The advantage of using the `server.xml` file for mapping roles to users is that your application does not need to be packaged into an EAR file and it is easier to update. Alternatively, using the `ibm-application-bnd.xmi/xml` file makes your application portable to other servers and other full profile servers that do not support the `server.xml` file.
3. If the required role is mapped to the **`EVERYONE`** special subject, then the authorization service returns immediately to allow anyone access. If the role is mapped to the **`ALL_AUHENTICATED`** special subject and the user is authenticated, then the authorization service grants access to the user. If none of these conditions are met, then the authorization service determines what users and groups are mapped to the required role. The authorization service grants access to the resource if the user is mapped to the required role or if the user is part of a group that is mapped to the role.
4. The authorization service returns a result back to the web container indicating whether the user is granted or denied access.

Special subjects

When mapping entities to roles, you can map a special subject, instead of a specific user or group. A special subject is an extension to the concept of a subject. A special subject can represent a group of users that fall under a specific category.

There are two types of special subjects:

- EVERYONE: represents any entity on the system, which means that there is no security because everyone is allowed access and you are not prompted to enter credentials.
- ALL_AUTHENTICATED_USERS: represents any entity that has successfully authenticated to the server.

In order to map a special subject to a user, you can update either the ibm-application-bnd.xmi/xml file of the application or the server.xml file. In this example, the role named AllAuthenticated is mapped to the special subject ALL_AUTHENTICATED_USERS:

```
<application-bnd>
       <security-role name="AllAuthenticated">
           <special-subject type="ALL_AUTHENTICATED_USERS" />
       </security-role>
   </application-bnd>
```

See Configuring authorization for applications on the Liberty profile.

Access Ids and authorization

When authorizing a user or group, the server needs a way to uniquely identify that user or group. The unique ID of the user and group serve this purpose and are used to build the authorization table. These IDs are determined by the user registry implementation: the unique user ID is the value of getUniqueUserId(), and the unique group IDis the value of getUniqueGroupId(). You can also choose to explicitly specify an access ID for the user or group in the authorization table. These explicit access IDs are used instead of what is returned by the user registry implementation. In order to specify an access ID in the ibm-application-bnd.xml/xmi or the server.xml file, use the **access-id** attribute for the user or group element.

In this example, an access ID is specified for the user **Bob** and the group **developers**:

```
<application-bnd>
       <security-role name="Employee">
           <user name="Bob" access-id="user:MyRealm/Bob"/>
           <group name="developers" access-id="group:myRealm/developers"/>
       </security-role>
</application-bnd>
```

Avoid trouble: Specifying access IDs is not encouraged because you must code the unique IDs and realm name within your code, which causes problems when you want to modify your user registry.

Liberty profile: Security public APIs

This topic provides an overview of the security public APIs in the Liberty profile.

The Liberty profile contains public APIs that you can use to implement security functions. The security public APIs in the Liberty profile are a subset of the full profile security public APIs. The main classes are WSSecurityHelper, WSSubject, and RegistryHelper. These classes contain a subset of the methods that are available in the full profile versions. There is also a new class WebSecurityHelper.

The following sections describes those main classes. There are also other classes such as UserRegistry, WSCredential, and other exception classes.

All the security public APIs supported by the Liberty profile are in the Java API documentation. The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

`WSSecurityHelper`

This class contains only the methods `isServerSecurityEnabled()` and `isGlobalSecurityEnabled()`. They both return true if any of the following features are enabled in the `server.xml` file:

- `appSecurity-1.0`
- `zosSecurity-1.0`

Otherwise, they return false. These methods are carried over from the full profile `WSSecurityHelper` class for compatibility.

Note:

- There are no cells in the Liberty profile, so there is no distinction in the Liberty profile between global security and server security. Therefore both methods return the same value.
- The method `revokeSSOCookies(javax.servlet.http.HttpServletRequest req,javax.servlet.http.HttpServletResponse res)` is not supported in the Liberty profile. Instead you can use the Servlet 3.0 logout function.
- The method `getLTPACookieFromSSOToken()` has been renamed to a new public API class - `WebSecurityHelper`.

`WSSubject`

This class provides utility methods for querying and setting the security thread context. All methods from the full profile `WSSubject` are supported in the Liberty profile.

Note: Java 2 security is not enabled in the Liberty profile, so the Java 2 security checks in `WSSubject` are not performed.

`RegistryHelper`

This class provides access to the `UserRegistry` object and trusted realm information. In the Liberty profile, it contains the following subset of the full profile methods:

```
public static UserRegistry getUserRegistry(String realmName) throws WSSecurityException
```

```
public static List<String> getInboundTrustedRealms(String realmName) throws
WSSecurityException
```

```
public static boolean isRealmInboundTrusted(String inboundRealm, String localRealm)
```

`WebSecurityHelper`

This class contains the renamed `getLTPACookieFromSSOToken()` method, which was moved from `WSSecurityHelper`:

```
public static Cookie getLTPACookieFromSSOToken() throws Exception
```

Security public API code examples

The following examples demonstrate how to use security public APIs in the Liberty profile to do a programmatic login and operate on the Subject.

- Example 1: create a Subject and use it for authorization
- Example 2: create a Subject and make it as the current Subject on the thread
- Example 3: get information of the current Subject on the thread

Example 1: create a Subject and use it for authorization

This example demonstrates how to use WSSecurityHelper, WSSubject, and UserRegistry to do a programmatic login to create a Java Subject, then perform an action and use that Subject for any authorization that is required.

Note: The following code uses WSSecurityHelper to check if security is enabled before doing further security processing. This check is used extensively because of the modular nature of the Liberty profile: If security is not enabled then the security runtime is not loaded. WSSecurityHelper is always loaded, even if security is not enabled.

```
import java.rmi.RemoteException;
import java.security.PrivilegedAction;

import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.security.CustomRegistryException;
import com.ibm.websphere.security.UserRegistry;
import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.WSSecurityHelper;
import com.ibm.websphere.security.auth.WSSubject;
import com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl;
import com.ibm.wsspi.security.registry.RegistryHelper;
public class myServlet {

 ...
 if (WSSecurityHelper.isServerSecurityEnabled()) {
  UserRegistry ur = null;
  try {
   ur = RegistryHelper.getUserRegistry(null);
  } catch (WSSecurityException e1) {
   // record some diagnostic info
   return;
  }
  String userid = "user1";
  String password = "user1password";
  try {
   if (ur.isValidUser(userid)) {
    // create a Subject, authenticating with
    // a userid and password
    CallbackHandler wscbh = new WSCallbackHandlerImpl(userid, password);
    LoginContext ctx;
    ctx = new LoginContext("WSLogin", wscbh);
    ctx.login();
    Subject subject = ctx.getSubject();
    // Perform an action using the Subject for
    // any required authorization
    WSSubject.doAs(subject, action);
   }
  } catch (CustomRegistryException e) {
   // record some diagnostic info
   return;
  } catch (RemoteException e) {
   // record some diagnostic info
   return;
  } catch (LoginException e) {
   // record some diagnostic info
   return;
  }
 }
 ...
 private final PrivilegedAction action = new PrivilegedAction() {
  @Override
  public Object run() {
   // do something useful here
   return null;
  }
 };

}
```

Example 2: create a Subject and make it the current Subject on the thread

The following example demonstrates how to use `WSSecurityHelper` and `WSSubject` to do a programmatic login to create a Java Subject, make that Subject the current Subject on the thread, and finally restore the original security thread context.

Note: The following code uses `WSSecurityHelper` to check if security is enabled before doing further security processing.

```
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.WSSecurityHelper;
import com.ibm.websphere.security.auth.WSSubject;
import com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl;
...
if (WSSecurityHelper.isServerSecurityEnabled()) {
 CallbackHandler wscbh = new WSCallbackHandlerImpl("user1", "user1password");
 LoginContext ctx;
 try {
  // create a Subject, authenticating with
  // a userid and password
  ctx = new LoginContext("WSLogin", wscbh);
  ctx.login();
  Subject mySubject = ctx.getSubject();
  Subject oldSubject = null;
  try {
   // Save a ref to the current Subject on the thread
   oldSubject = WSSubject.getRunAsSubject();
   // Make mySubject the current Subject on the thread
   WSSubject.setRunAsSubject(mySubject);
   // Do something useful here. Any authorization
   // required will be performed using mySubject
  } catch (WSSecurityException e) {
   // record some diagnostic info
   return;
  } finally {
   // Put the original Subject back on the thread context
   if (oldSubject != null) {
    try {
     WSSubject.setRunAsSubject(oldSubject);
    } catch (WSSecurityException e) {
     // record some diagnostic info
    }
   }
  }
 } catch (LoginException e) {
  // record some diagnostic info
  return;
 }
}
```

Example 3: get information of the current Subject on the thread

The following example demonstrates how to use `WSSecurityHelper`, `WSSubject` and `WSCredential` to get information about the current Subject on the thread.

Note: The following code uses `WSSecurityHelper` to check if security is enabled before doing further security processing.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.login.CredentialExpiredException;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.WSSecurityHelper;
import com.ibm.websphere.security.auth.CredentialDestroyedException;
import com.ibm.websphere.security.auth.WSSubject;
import com.ibm.websphere.security.cred.WSCredential;
...
if (WSSecurityHelper.isServerSecurityEnabled()) {
 // Get the caller's subject
 Subject callerSubject;
 try {
  callerSubject = WSSubject.getCallerSubject();
 } catch (WSSecurityException e) {
  // record some diagnostic info
  return;
 }
 WSCredential wsCred = null;
 Set<WSCredential> wsCredentials =
  callerSubject.getPublicCredentials(WSCredential.class);
 Iterator<WSCredential> wsCredentialsIterator = wsCredentials.iterator();
 if (wsCredentialsIterator.hasNext()) {
  wsCred = wsCredentialsIterator.next();
  try {
   // Print out the groups
   ArrayList<String> groups = wsCred.getGroupIds();
   for (String group : groups) {
    System.out.println("Group name: " + group);
   }
  } catch (CredentialExpiredException e) {
   // record some diagnostic info
   return;
  } catch (CredentialDestroyedException e) {
   // record some diagnostic info
   return;
  }
 }
}
}
```

Configuration differences between the full profile and Liberty profile: security

This topic describes the main differences in the security capability between the Liberty profile and full profile.

The Liberty profile security supports only a subset of security features in the full profile. Unless the support is explicitly mentioned in the Liberty profile documentation, you have to assume that the support is not available yet.

Some of the main security features missing in the Liberty profile are:

- No Java EE security.
- Not all public APIs and SPIs are supported. The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.
- No custom user registry.
- No horizontal propagation.

- No SecurityAdmin MBean support, therefore methods like clearing the authentication cache are not available.
- No Java Authorization Contract for Container (JACC) support.
- No Java 2 Connector (J2C) principal mapping modules support.
- No Java Authentication SPI (JASPI) support.
- No multiple security domain support.
- No security auditing subsystem that is part of the security infrastructure of the server.

In the Liberty profile, you can configure user-to-role mappings and runAs users in the application-bnd element of the server.xml file. In the full profile, you can only configure this in the ibm-application-bnd.xml/xmi file. See Configuring authorization for applications on the Liberty profile.

Chapter 13. Mail, URLs, and other Java EE resources

This page provides a starting point for finding information about resources that are used by applications that are deployed on a Java Enterprise Edition (Java EE)-compliant application server. They include:

- JavaMail support for applications to send Internet mail
- URLs, for describing logical locations
- Resource environment entries, for mapping logical names to physical names
- Java DataBase Connectivity (JDBC) resources and other technology for data access (discussed elsewhere)
- Java Message Service (JMS) resources and other messaging system support (discussed elsewhere)

Mail service providers and mail sessions

A mail service provider is a driver that supports mail interaction with mail servers that use a particular mail protocol. The application server includes service providers, which are also known as protocol providers, for mail protocols.

A mail provider encapsulates a collection of protocol providers. For example, the application server has a built-in mail provider that encompasses the most common protocol providers. These protocol providers are installed as the default and suffice for most applications. If you have a particular application that requires custom protocol providers, follow the steps that are outlined in the chapter on mail sessions in the JavaMail API Design Specification to install your own protocol providers.

Mail sessions are represented by the javax.mail.Session class. A mail session object authenticates users and controls access to messaging systems.

To create mail applications that are platform independent, use a resource factory reference to create a mail session. A resource factory is an object that provides access to resources in the deployed environment of a program. Resource factories use the naming conventions that are defined by the Java Naming and Directory Interface (JNDI).

Note: Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.

Mail: Resources for learning

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

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

Programming model and decisions

- JavaMail documentation

Programming specifications

- JavaMail 1.3 API documentation (Sun Java specifications)

JavaMail support for Internet Protocol 6.0

WebSphere Application Server and its JavaMail component support Internet Protocol Version 6.0 (IPv6).

Support for IPv6, includes the following:

- Both can run on a pure IPv4 network, a pure IPv6 network, *or* a mixed IPv4 and IPv6 network.
- On either the pure IPv6 network or the mixed network, the JavaMail component works with mail servers, such as the SMTP mail transfer agent, and the IMAP and POP3 mail stores.
- IPv6 compatible. Additionally, a JavaMail component that is run on the mixed IPv4 and IPv6 network can communicate with mail servers using IPv4.

When you configure a mail session, you can specify the mail server hosts (also known as mail transport and mail store hosts) with domain-qualified host names or numerical IP addresses. Using host names is generally the preferred method. If you use IP addresses, however, consider enclosing IPv6 addresses in square brackets to prevent parsing inaccuracies. For example:

```
[fe80::202:57ff:fec4:2334]
```

The JavaMail API requires a combination of many host names or IP addresses with a port number, using the `host:port number` syntax . This extra colon can cause the port number to be read as part of an IPv6 address. Using brackets prevents your JavaMail implementation from processing the extra characters erroneously.

URLs

A Uniform Resource Locator (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as HTTP, FTP, file, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The *scheme_information* for HTTP, FTP and file generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

```
http://www.ibm.com/software/webservers/appserv/library.html.
```

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URLs: Resources for learning

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

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

Programming specifications

- W3C Architecture - Naming and Addressing: URIs, URLs
- URL API documentation

Chapter 14. Managed beans

This page provides a starting point for finding information about Managed beans.

Managed beans are container-managed objects with minimal requirements, otherwise known as Plain Old Java Objects (POJO). They support a small set of basic services, such as resource injection, life cycle callbacks, and interceptors. Other, more advanced, aspects are introduced in companion specifications, to keep the basic model as simple and as universally useful as possible.

Managed beans offer a lightweight component model aligned with the rest of the Java Platform Enterprise Edition (Java EE). By supporting the common resource injection and life cycle services, Managed beans fit into the Java EE programming model. At the same time, the lightweight nature of Managed beans makes them a natural starting point to encapsulate application functionality, with the knowledge that they can be formed into more powerful components. In this sense, Managed beans can be seen as a Java EE platform-enhanced version of the JavaBeans component model found on the Java Platform Standard Edition (Java SE).

Managed beans

The Managed Beans specification (JSR -316) is used to define managed beans for the Java Platform Enterprise Edition (EE) and is a part of the Java EE 6 platform.

Managed beans are container-managed objects with minimal supported services, such as resource injection, life cycle callbacks and interceptors, and have the following characteristics:

- A managed bean does not have its own component-scoped java:comp namespace. Therefore, its resources can be defined in java:app and java:module only.
- Managed beans are local beans only and cannot be defined in java:global.
- Managed bean methods run in the same thread as the calling thread. For example, the method does not start its own thread.
- Managed bean methods use the same context as the calling thread.
- Managed beans are defined with the javax.annotation.ManagedBean annotation.
- A managed bean can have an optional name and is bound into java:module and java:app only if a name is present; for example:

  ```
  @ManagedBean("myCart")
  public class Cart { ... }
  ```
- A reference to a managed bean can be obtained through resource injection, or lookup in java:module or java:app, when a name is specified.
- Managed beans support the PostConstruct and PreDestroy life cycle callbacks.

Chapter 15. Messaging resources

This page provides a starting point for finding information about the use of asynchronous messaging resources for enterprise applications with WebSphere Application Server.

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and the Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with traditional WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider).
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider).
- Third-party messaging providers that implement either a JCA Version 1.5 or 1.6 resource adapter or the ASF component of the JMS Version 1.0.2 specification.

Styles of messaging in applications

Applications can use point-to-point and publish/subscribe messaging. These styles of messaging can be used in the following ways: one-way; request and response; one-way and forward.

Applications can use the following styles of asynchronous messaging:

Point-to-point

Point-to-point applications typically use *queues* to pass messages between each other. An application sends a message to another application by identifying, implicitly or explicitly, a destination queue. The underlying messaging and queuing system receives the message from the sending application and routes the message to its destination queue. The receiving application can then retrieve the message from the queue.

Publish/subscribe

In publish/subscribe messaging, there are two types of application: publisher and subscriber.

A *publisher* supplies information in the form of messages. When a publisher publishes a message, it specifies a *topic*, which identifies the subject of the information inside the message.

A *subscriber* is a consumer of the information that is published. A subscriber specifies the topics it is interested in by sending subscription requests to a publish/subscribe broker. The broker receives published messages from publishers and subscription requests from subscribers, and it routes published messages to subscribers. A subscriber receives messages on only those topics to which it has subscribed.

Both styles of messaging can be used in the same application.

Applications can use asynchronous messaging in the following ways:

One-way

An application sends a message, and does not want a response. A message like this can be referred to as a *datagram*.

One-way and forward

An application sends a request to another application, which sends a message to yet another application.

Request and response

An application sends a request to another application and expects to receive a response in return.

A typical JMS messaging pattern involves a requesting application sending a message to a JMS queue for processing by a messaging service (for example, a message-driven bean). When the requesting application sends the request message, the message identifies another JMS queue to which the service should send a reply message. After sending the request message, the requesting application either waits for the reply message to arrive, or it reconnects later to retrieve the reply message.

These messaging techniques can be combined to produce a variety of asynchronous messaging scenarios.

For details of how WebSphere applications can use JMS and message-driven beans for asynchronous messaging, see the following topics:
- "JMS interfaces - explicit polling for messages" on page 458
- "Message-driven beans - automatic message retrieval" on page 142

For more information about these messaging techniques and the Java Message Service (JMS), see Sun's Java Message Service (JMS) specification documentation (http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/).

For more information about message-driven bean and inbound messaging support, see Sun's Enterprise JavaBeans specification (http://java.sun.com/products/ejb/docs.html).

For information about JCA inbound messaging processing, see Sun's J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).

Types of messaging providers

You can configure any of three main types of Java Message Service (JMS) providers in WebSphere Application Server: The WebSphere Application Server default messaging provider (which uses service integration as the provider), the WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider) and third-party messaging providers (which use another company's product as the provider).

Overview

WebSphere Application Server supports JMS messaging through the following providers:
- "Default messaging provider" on page 349
- "WebSphere MQ messaging provider" on page 349
- "Third-party messaging provider" on page 350

Your applications can use messaging resources from any of these JMS providers. The choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you might already have a messaging infrastructure based on WebSphere MQ. In this case, you can either connect directly by using the WebSphere MQ messaging provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

You can have more than one type of messaging provider configured in WebSphere Application Server:
- All types of provider can be configured within one cell.

- Different applications can use the same, or different, providers.
- One application can access multiple providers.

Default messaging provider

If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is a logical choice. This provider uses service integration functions and is part of the WebSphere Application Server runtime environment.

To use the default messaging provider, your applications connect to a service integration bus. You can assign JMS queues (for point-to-point messaging) or JMS topics (for publish/subscribe messaging) as destinations on the service integration bus.

The default messaging provider is characterized as follows:

- A service integration bus comprises messaging engines that run in WebSphere Application Server processes and dynamically connect to one another by using dynamic discovery. A messaging application connects to the bus through a messaging engine.
- Messaging engines use WebSphere Application Server clustering to provide high availability and scalability, and they use the same management framework as the rest of WebSphere Application Server.
- Bus client applications can run from within WebSphere Application Server (JMS), or run as stand-alone Java clients (using the J2SE Client for JMS) or run as non-Java clients (XMS).

There are two ways in which you can connect to a WebSphere MQ system through the default messaging provider:

- Connect a bus to a WebSphere MQ network, by using a *WebSphere MQ link*. The WebSphere MQ network appears to the service integration bus as a foreign bus, and the service integration bus appears to WebSphere MQ as another queue manager.
- Connect directly to WebSphere MQ queues located on WebSphere MQ queue managers or (for WebSphere MQ for z/OS) queue-sharing groups, by using a *WebSphere MQ server* bus member. Each WebSphere MQ queue is made available at a queue-type destination on the bus.

For more information about these two approaches, see “Interoperation with WebSphere MQ: Comparison of key features” on page 376.

To configure and manage messaging with the default messaging provider, see Managing messaging with the default messaging provider.

WebSphere MQ messaging provider

Through the WebSphere MQ messaging provider in WebSphere Application Server, Java Message Service (JMS) messaging applications can use your WebSphere MQ system as an external provider of JMS messaging resources.

You can use WebSphere Application Server to configure WebSphere MQ resources for applications (for example queue connection factories) and to manage messages and subscriptions associated with JMS destinations. You administer security through WebSphere MQ.

WebSphere MQ is characterized as follows:

- Messaging is handled by a network of queue managers, each running in its own set of processes and having its own administration.
- Features such as shared queues (on WebSphere MQ for z/OS) and WebSphere MQ clustering simplify administration and provide dynamic discovery.

- Many IBM and partner products support WebSphere MQ with (for example) monitoring and control, high availability and clustering.
- WebSphere MQ clients can run within WebSphere Application Server (JMS), or almost any other messaging environment by using a variety of APIs.

For more information about the WebSphere MQ messaging provider, see “Interoperation using the WebSphere MQ messaging provider” on page 382. To configure and manage messaging with this provider, see Managing messaging with the WebSphere MQ messaging provider.

Third-party messaging provider

You can configure any third-party messaging provider that supports the JMS Version 1.1 specification. You might want to do this, for example, if you have existing investments.

To administer a third-party messaging provider, you use either the resource adaptor (for a Java EE Connector Architecture (JCA) 1.5-compliant or 1.6-compliant messaging provider) or the client (for a non-JCA messaging provider) that is supplied by the third party. You use the WebSphere Application Server administrative console to administer the activation specifications, connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

To use message-driven beans, third-party messaging providers must either provide an inbound JCA 1.5-compliant or 1.6-compliant-resource adapter, or (for non-JCA messaging providers) include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification.

To work with a third-party provider, see Managing messaging with a third-party JCA 1.5 or 1.6-compliant messaging provider or Managing messaging with a third-party non-JCA messaging provider.

Default messaging

Use these topics to learn about using the default messaging provider to support the use of the Java Message Service (JMS) by enterprise applications deployed on WebSphere Application Server.

The default messaging provider is installed and runs as part of WebSphere Application Server.

The default messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as “unified” or “common” interfaces). This enables applications to use the same common interfaces for both point-to-point and publish/subscribe messaging. This also enables both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, this approach is recommended for new applications. The domain-specific interfaces are supported for backwards compatibility as described in section 1.5 of the JMS 1.1 specification.

The default messaging provider is based on service integration technologies. You can use the WebSphere Application Server administrative console to configure JMS resources:

- A JCA activation specification to enable a message-driven bean to communicate with the default messaging provider.
- A JMS connection factory to connect to a service integration bus
- A JMS queue or topic assigned to a bus destination on the bus. Such JMS queues and topics are available, over a long period of time, to all applications with access to the bus destination.

For more information about using the default messaging provider to support JMS messaging, see the following topics:

- “JCA activation specifications and service integration” on page 351

- “JMS connection factories and service integration”
- “JMS queue resources and service integration” on page 352
- “JMS topic resources and service integration” on page 353
- “The createQueue or createTopic method and the default messaging provider” on page 355
- “How JMS applications connect to a messaging engine on a bus” on page 358

JCA activation specifications and service integration

A Java EE Connector Architecture (JCA) 1.5 activation specification enables a message-driven bean to communicate with the default messaging provider.

You create a JMS activation specification if you want to use a message-driven bean to communicate with the default messaging provider through Java EE Connector Architecture (JCA) 1.5. JCA provides Java connectivity between application servers such as WebSphere Application Server, and enterprise information systems. It provides a standardized way of integrating JMS providers with Java EE application servers, and provides a framework for exchanging data with enterprise systems, where data is transferred in the form of messages.

One or more message-driven beans can share a single JMS activation specification.

All the activation specification configuration properties apart from **Name**, **JNDI name**, **Destination JNDI name**, and **Authentication alias** are overridden by appropriately named activation-configuration properties in the deployment descriptor of an associated EJB 2.1 or later message-driven bean. For an EJB 2.0 message-driven bean, the **Destination type**, **Subscription durability**, **Acknowledge mode** and **Message selector** properties are overridden by the corresponding elements in the deployment descriptor. For either type of bean the **Destination JNDI name** property can be overridden by a value specified in the message-driven bean bindings.

JMS connection factories and service integration

A JMS connection factory is used to create connections to JMS resources on a service integration bus.

A “domain-independent” JMS connection factory supports the JMS 1.1 domain-independent interfaces (sometimes referred to as the “unified” or “common” interfaces). This enables applications to use the same, common, interfaces for both point-to-point and publish/subscribe messaging. This also enables both point-to-point and publish/subscribe messaging within the same transaction.

Due to the interface inheritance defined by the JMS specification, a JMS 1.1 application can use a JMS 1.0.2b, domain-specific, connection factory. However, a JMS 1.0.2b application cannot use a JMS 1.1 domain-independent connection factory.

You should use the connection factory type that matches the JMS level and domain pattern in which an application is developed. For example, use a domain-independent JMS connection factory for a JMS application developed to use JMS 1.1 domain-independent interfaces, and use a JMS queue connection factory for a JMS application developed to use domain-specific queue interfaces.

Applications running in a server that is a member of a bus can locate a messaging engine in that bus. Client applications running outside of an application server - for example, running in a client container or outside the WebSphere Application Server environment - cannot locate directly a suitable messaging engine to connect to in the target bus. Similarly, an application running on a server in one cell to connect to a target bus in another cell cannot locate directly a suitable messaging engine to connect to in the target bus.

In these scenarios, the clients (or servers in another bus) must complete a bootstrap process through a *bootstrap server* that is a member of the target bus. A bootstrap server is an application server running the SIB Service, but does not have to be running any messaging engines. The bootstrap server selects a messaging engine that is running in an application server that supports the required *target transport chain*. For the bootstrap process to be possible, you must configure one or more provider end points in the connection factory used by the client.

JMS queue resources and service integration

JMS queue resources (queues and queue connection factories) are provided by the default messaging provider for JMS point-to-point messaging and supported by a service integration bus.

The figure later in this section shows a bus with two members, a server and cluster. The two members each have a JMS queue. An application sends messages to one JMS queue and retrieves messages from the other JMS queue. There are queue destinations on a service integration bus and the JMS connection factories. These objects are described in detail below.

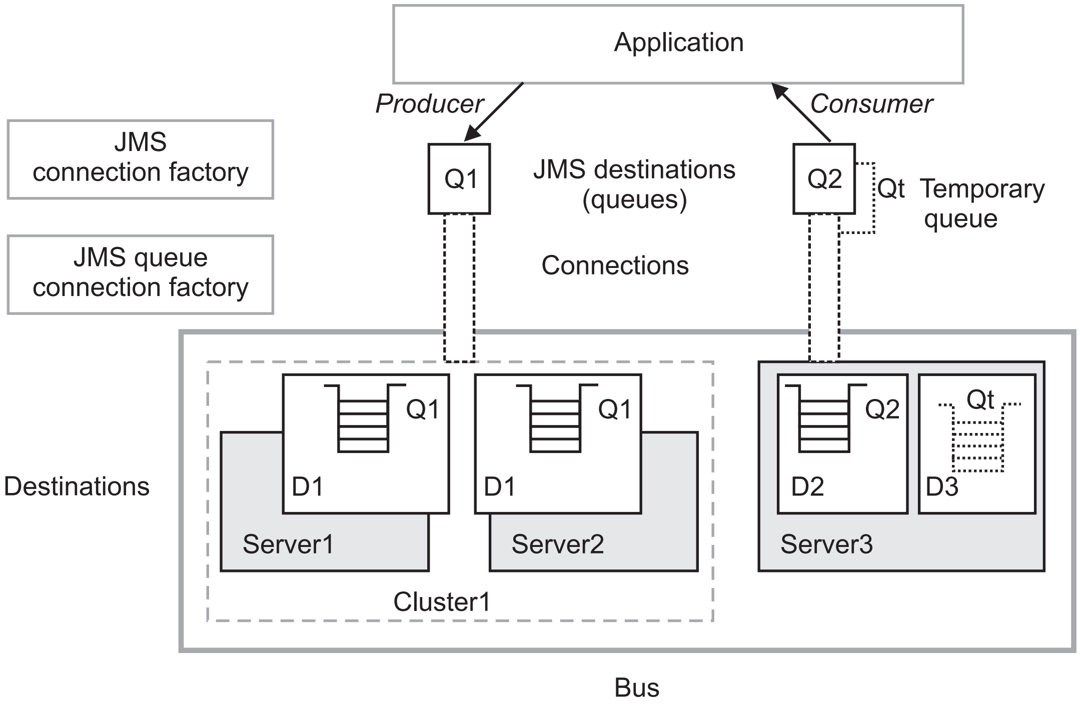


Figure 22. JMS point-to-point messaging and the default messaging provider

JMS queue

The term "JMS queue" is used to refer to the JMS destination (an instance of javax.jms.Queue) that applications interact with, and that an administrator configures as a JMS resource of the default messaging provider.

An administrator can define a *JMS queue*, an administrative object that encapsulates the name of a queue destination on a service integration bus. Applications can obtain the JMS queue by looking its name up in the JNDI namespace.

Applications that uses JMS point-to-point messaging act as producers or consumers of messages with JMS queues, and have no need to know about the service integration resources that support JMS queues.

Queue

The term "queue" is used as an abbreviation for "queue destination", and refers to a service integration bus destination configured for point-to-point messaging.

The administrator assigns the queue to only one member (an application server) of the bus. The messaging engine in the bus member hosts the message point for the queue, known as a queue point. The queue point is the location where messages for the queue are stored and processed on the bus.

If the bus member has more than one messaging engine, the queue is partitioned across the messaging engines. Each messaging engine hosts a separate queue point for the queue.

JMS connection factory

A “JMS connection factory” creates connections to a messaging engine through which it can access messages on queue points anywhere on the bus.

With JMS 1.1, you are recommended to use domain-independent JMS connection factories for new applications. Domain-specific queue connection factories are supported for backwards compatibility for JMS applications developed to use domain-specific queue interfaces, as described in section 1.5 of the JMS 1.1 specification.

Temporary JMS queues

In addition to using JMS queues that are created as administrative objects, an application can also create its own temporary JMS queues, which exist at runtime only for the duration of a connection. Only that connection can create MessageConsumers for the temporary JMS queue; for example, for use as the JMSReplyTo queue for service requests.

For more information about creating temporary JMS destinations, see section 4.43 of the JMS 1.1 specification.

For a temporary JMS queue, the service integration bus creates a temporary destination, which the administrator can list and browse but usually does not have to act on.

JMS topic resources and service integration

JMS topic resources (topics, topic spaces, connection factories, durable subscriptions) are provided by the default messaging provider for JMS publish/subscribe messaging, and supported by a service integration bus.

JMS publish/subscribe messaging and the default messaging provider is shown in the following figure:

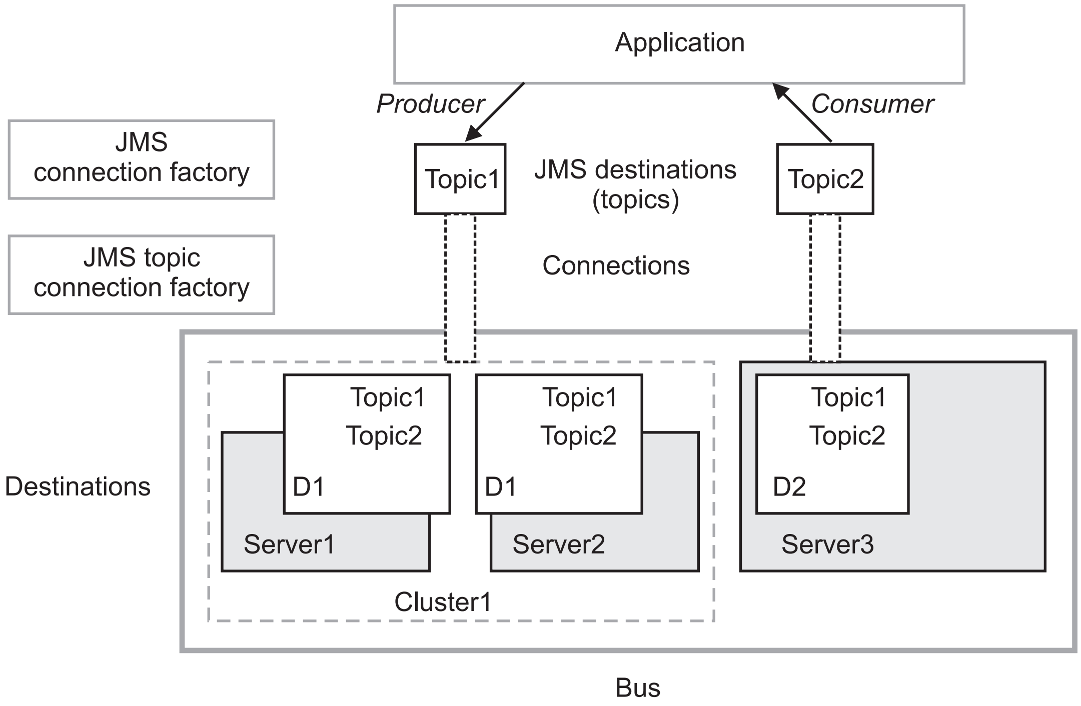


Figure 23. JMS publish/subscribe messaging and the default messaging provider

JMS topic

The term “JMS topic” is used to refer to the JMS destination (an instance of javax.jms.Topic) that applications interact with, and that an administrator configures as a JMS resource of the default messaging provider.

An application that uses JMS publish/subscribe messaging acts as a producer or consumer of messages with JMS topics, and has no need to know about other service integration resources that support the JMS topic.

An administrator can define a *JMS topic*, an administrative object that encapsulates the name of a topic and a topic space on a service integration bus. Applications can obtain the JMS topic by looking its name up in the JNDI namespace.

JMS applications can publish messages to, and subscribe to messages from, JMS topics. Subscribing applications can usually receive messages published to a topic only when the subscriber is connected to the server.

The default messaging provider also supports the use of durable subscriptions to topics, which enable the subscriber to receive messages that were published when the subscriber was disconnected. For more information about durable subscriptions, see section 6.11.1 of the JMS 1.1 specification.

Topic space

A *topic space* (a hierarchical collection of topics) is a virtual location on a service integration bus where messages are stored and processed for publish/subscribe messaging.

Unlike configuring queues, the administrator does not have to assign the topic space to a bus member. A topic space has a *publication point* defined automatically for each messaging engine in the bus. Messages for the topic space are stored and processed on all its publication points.

Topic The term “topic” refers to a discriminator within a topic space.

When subscribing to topics, applications can specify wildcard characters to select a range of topics.

JMS connection factory

A “JMS connection factory” creates connections to a messaging engine that provides a publication point for the topic space.

With JMS 1.1, you are recommended to use domain-independent JMS connection factories for new applications. Domain-specific topic connection factories are supported for backwards compatibility for JMS applications developed to use domain-specific topic interfaces, as described in section 1.5 of the JMS 1.1 specification.

Temporary JMS topics

In addition to using JMS topics that are created as administrative objects, an application can also create its own temporary JMS topics, which exist at runtime only for the duration of a connection. Only that connection can create MessageConsumers for the temporary JMS topic.

For more information about creating temporary JMS destinations, see section 4.43 of the JMS 1.1 specification.

For a temporary JMS topic, the service integration bus creates a temporary topic space, which the administrator can list and browse but usually does not have to act on. A temporary topic space is deleted automatically when the connection is closed.

Durable subscriptions

A durable subscription on a JMS topic enables a subscriber to receive a copy of all messages published to that topic, even messages published during periods of time when the subscriber is not connected to the server. Therefore, subscriber applications can operate disconnected from the server for long periods of time, and then reconnect to the server and process messages that were published during their absence. If an application creates a durable subscription, it is added to the list that administrators can display and act on by using the administrative console.

The createQueue or createTopic method and the default messaging provider

You can use the Session.createQueue(String) method or Session.createTopic(String) method instead of using JNDI lookup to create a JMS Queue or JMS Topic with the default messaging provider.

Applications can use the InitialContext.lookup() method to retrieve administered objects. An alternative, but less manageable, approach to obtaining administratively defined JMS destination objects by JNDI lookup is to use the Session.createQueue(String) method or Session.createTopic(String) method. For example,

```
Queue q = mySession.createQueue("Q1");
```

creates a JMS Queue instance that can be used to reference the existing destination Q1.

With the default messaging provider, the existing destination exists as a queue or topic space on the bus to which the session is connected.

createQueue

The Session.createQueue(String) method is used to create a JMS Queue object representing an existing destination. This provides an alternative, but less manageable, approach to obtaining administratively-defined JMS Queue objects by JNDI lookup.

Simple form

In its simplest form, the parameter to the createQueue method is the name of an existing destination on the bus to which the session is connected. For example, if there exists a queue named Q1 then the following method creates a JMS Queue instance that can be used to reference that destination:

```
Queue q = mySession.createQueue("Q1");
```

URI form

For more complex situations, applications can use a URI-based format. The URI format allows an arbitrary number of name value pairs to be supplied to set various properties of the Queue object. The queue URI is identified by the prefix `queue://`, followed by the name of the destination. The simple form for Q1 previously, can be expressed with the following URI:

```
Queue q = mySession.createQueue("queue://Q1");
```

Name value pairs are introduced by a question mark `?`. For example, an application might connect a session to one bus then use the following format to create a JMS Queue instance for Q2 on a different bus, called otherBus:

```
Queue q = mySession.createQueue("queue://Q2?busName=otherBus");
```

When sending messages to WebSphere MQ, the queue name must be followed by an at sign (@) and the name of the queue manager on which the queue is located, for example :

```
Queue q = mySession.createQueue("queue://Q2@qmgr?busName=otherBus");
```

Multiple name value pairs are separated by an ampersand character `&`, for example:

```
Queue q = mySession.createQueue("queue://Q2?busName=otherBus&deliveryMode=
Application&readAhead=AsConnection&priority=6");
```

Properties

busName, **deliveryMode**, **priority**, **readAhead**, and **timeToLive**. See the generated API information for a description of these properties.

createTopic

The Session.createTopic(String) method is used to create a JMS Topic object representing an existing destination. (Note that for topics it is the topic space rather than the topic that exists.) This provides an alternative, but less manageable, approach to obtaining administratively-defined JMS Topic objects by JNDI lookup.

Simple form

In its simplest form, the parameter to the createTopic method is the name of a topic in the default topic space on the bus to which the session is connected. For example, if the default topic space exists, then a JMS Topic instance that can be used to reference the `cats` topic on the default topic space:

```
Topic t = mySession.createTopic("cats");
```

To specify a non-default topic space, the special syntax of the form `topicSpace:topic` can be used. For example:

```
Topic t = mySession.createTopic("kennelTopicSpace:dogs");
```

URI form

For more complex situations a URI based format can be used. The topic URI is identified by the prefix `topic://` followed by the name of the topic. The previous examples can be expressed as the following URIs:

```
Topic t = mySession.createTopic("topic://cats");

Topic t = mySession.createTopic("topic://dogs?topicSpace=kennelTopicSpace");
```

As for queues, multiple name value pairs are separated by an ampersand `&`.

Properties

busName, **deliveryMode**, **priority**, **readAhead**, **timeToLive**, and **topicSpace**. See the generated API information for a description of these properties.

Support for MA88 URIs

WebSphere Application Server Version 5.1 applications can use createQueue and createTopic methods to create JMS Queue and Topic objects with the Version 5 embedded messaging provider (the Version 5.1 JMS messaging provider). To assist you in migrating these applications, the default messaging provider (the service integration bus) supports a large subset of valid MA88-specific string parameters to the createQueue and createTopic methods.

Default queue manager

An MA88 URI for a queue includes the name of the queue manager; for example:

```
queue://qm/queue
```

To specify the default queue manager, the queue manager name is omitted; for example: `queue:///queue` (note the three forward slash characters, `///`). Because the interpretation of the default queue manager is logically consistent with the concept of a queue on the current bus, the bus tolerates the presence of three forward slash characters following the `queue:` prefix. This allows MA88 queue URIs with a default queue manager to be used by the bus without change.

Non-default queue manager

If an MA88 queue URI specifies a non-default queue manager, as in `queue://qm/queue`, then this has an ambiguous interpretation in the bus. To highlight the potential problem and ensure that the destination is given consideration during the porting process, such a URI generates a JMSException if passed to the createQueue() method.

MA88 properties

As with the bus URIs, MA88 URIs can contain a number of name value pairs specifying destination properties. Many of the MA88 specific properties have no direct equivalent in the bus and are ignored silently. However, the following MA88 properties are mapped to bus equivalents:

MA88 name	Service integration bus name	Notes
`expiry`	**`timeToLive`**	
`persistence`	**`deliveryMode`**	1 = NonPersistent 2 = Persistent Anything else = Application

Topic wildcard translation

A topic used for consuming messages can include wild cards. The wild card syntax used in MA88 differs from the XPath syntax used in the bus, so if an MA88 URI contains wild cards the bus attempts to convert them to XPath equivalents. The conversion performed depends on the presence of the **`brokerVersion`** property in the MA88 URI. The WebSphere Application Server Version 5.1 default messaging provider required any URI specifying a topic wild card to include `brokerVersion=1` in the name value pairs. The bus therefore uses `brokerVersion=1` as the trigger to undertake MQSI to XPath wild card conversion.

Case sensitivity

All parts of the string parameter for createQueue and createTopic are case sensitive.

Multiple instances of same property

If a URI contains multiple occurrences of a given property with conflicting values, it is not specified which value is used.

Conflicting MA88 and bus properties

If a URI contains both a property and the MA88 equivalent of that property with conflicting values, it is not specified which value is used.

Unknown properties

Any name value pairs for which the property name is not recognized are ignored without any error reporting.

Escaping special characters

The following characters have special significance in the createQueue and createTopic string parameters:

: (colon)
This is used as a separator between the topic space and the topic in short form topic strings

? (question mark)
This is used to indicate the start of the name value pairs.

& (ampersand)
This is used to separate multiple name value pairs.

If you want to use any of these characters in a URI, you must prefix it with a backward slash \. The \ character can also be escaped by doubling it; \\. Note that the \ character is treated as a special character by the Java language, and so must be doubled when placed in character string constants; for example:

```
createTopic("myTop\\:ic")                   creates a topic with the name "myTop:ic"
createTopic("topic://my\\?Topi\\\\c")        creates a topic with the name "my?Topi\c"
createQueue("queue://q1?busName=silly\\&bus")  creates a queue with bus name "silly&bus"
```

How JMS applications connect to a messaging engine on a bus

There are several factors that affect how JMS applications connect to a service integration bus, so that they can use resources provided by the bus.

To connect to a service integration bus, an application actually connects to a messaging engine on the bus.

By default, the environment automatically connects applications to an available messaging engine on the bus. However you can specify extra configuration details to influence the connection process; for example to identify special bootstrap servers, or to limit connection to a subgroup of available messaging engines, or to improve availability or performance, or to ensure sequential processing of messages received.

Applications running in an application server: Default configuration

Applications that are running in an application server are directed by the WebSphere Application Server environment to an available messaging engine.

If the messaging engine is found in the same server, a connection is created that provides the application with the fastest available connection to a messaging engine. Otherwise, if a messaging engine is found in another process - on the same or a different host - a remote connection is made. If no suitable messaging engine is found the application fails to connect to the bus.

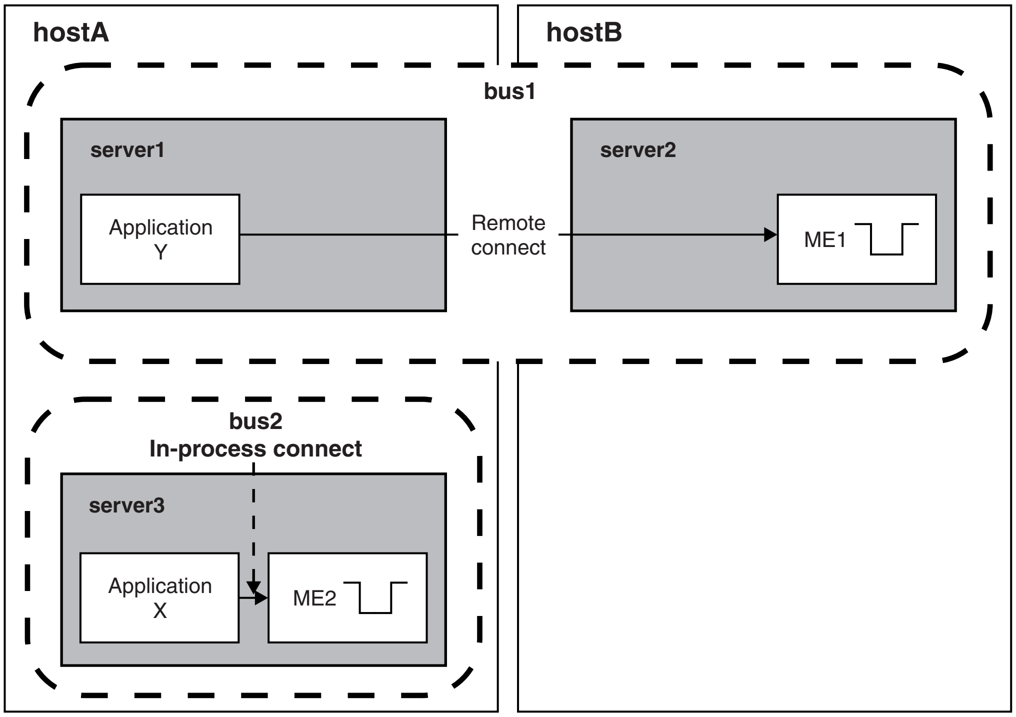


Figure 24. Default connection to a messaging engine - Applications running in an application server

The figure shows two applications running in application servers. Application X on server3 has connected to the messaging engine running in the same server. Application Y on server1 has connected to a messaging engine that is running in the same bus but on a different server and host, because server1 does not have a suitable messaging engine.

Applications running outside an application server

Client applications running outside an application server (for example, running in a client container or outside the WebSphere Application Server environment) cannot locate a suitable messaging engine themselves and must complete a bootstrap process through a *bootstrap server*. A bootstrap server is an application server that is running the SIBService service, but is not necessarily running any messaging engines. The bootstrap server selects a messaging engine that is running in an application server that supports the required *target transport chain*.

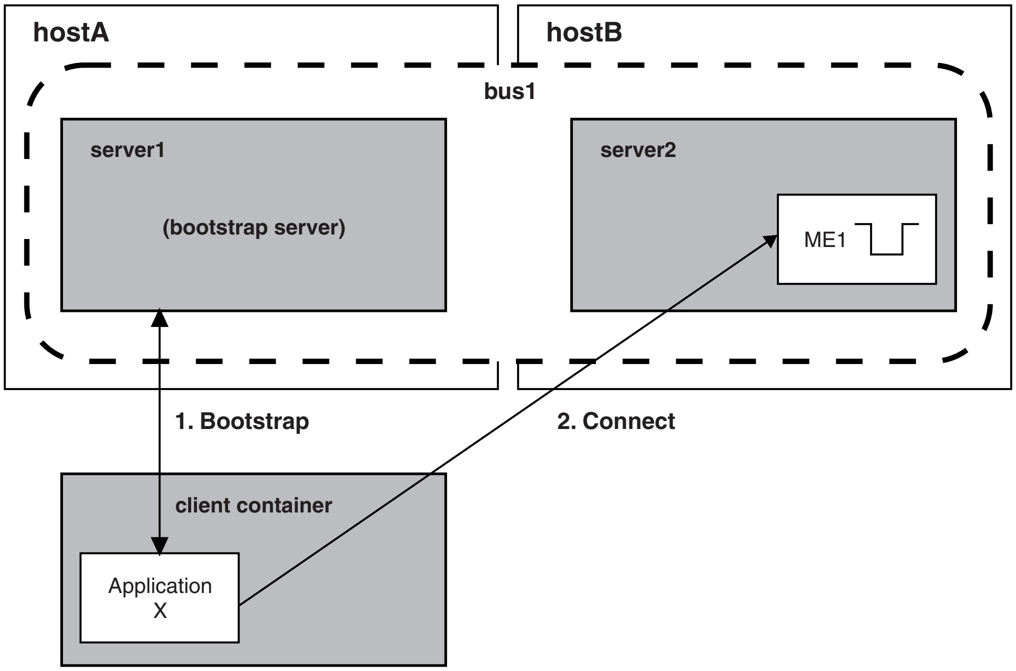


Figure 25. Connection to a messaging engine - Applications running outside an application server

This figure shows a client application running outside an application server. To connect to a messaging engine, the application connects first to a bootstrap server. The bootstrap server selects a messaging engine then tells the client application to connect to that messaging engine.

A bootstrap server uses a specific port and bootstrap transport chain, which with the host name form the *endpoint address* of the bootstrap server.

The properties of a JMS connection factory used by a client application control the selection of a suitable messaging engine and how the client connects to the selected messaging engine. By default, a connection factory expects to use a bootstrap server that has an endpoint address of `localhost:7276:BootstrapBasicMessaging`. That is: the client application expects to use a bootstrap server that is on the same host as the client, and that uses port 7276 and the predefined bootstrap transport chain called BootstrapBasicMessaging.

Note: IBM i For the IBM i platform, you must (at least) change the default endpoint address from `localhost:7276:BootstrapBasicMessaging` to *`your.server.name`*`:7276:BootstrapBasicMessaging`.

When you create an application server, it is automatically assigned a unique non-secure bootstrap port, SIB_ENDPOINT_ADDRESS, and a secure bootstrap port, SIB_ENDPOINT_SECURE_ADDRESS. If you want to use an application server as a bootstrap server, and the server has been assigned a non-secure port other than 7276, or you want to use the secure port, then you must specify the endpoint address of the server on the Provider endpoints property of the connection factory.

The endpoint addresses for bootstrap servers must be specified in every connection factory that is used by applications outside of an application server. To avoid having to specify a long list of bootstrap servers, you can provide a few highly-available servers as dedicated bootstrap servers. Then you only have to specify a short list of bootstrap servers on each connection factory.

The messaging engine selection process

The selection process is used to choose a messaging engine that an application should connect to so that it can use the resources of a service integration bus. The information that controls the selection process is configured in one of the following places:

- For JMS client applications, this information is configured on the connection factory.
- For message-driven bean (MDB) applications, this information is configured on the activation specification.
- For other types of application, this information is configured programmatically by the application.

Although a connection can be made to any available messaging engine, the connection process applies a few simple rules to find the most suitable messaging engine. For an application running in an application server, the process is as follows:

1. If a messaging engine is running in the required bus within the same application server, then a connection is made from the application to the messaging engine. If there is no suitable messaging engine, the next rule is checked.
2. If a messaging engine is running on the same host as the application, then the application makes a remote connection to the selected messaging engine. If there is no suitable messaging engine, the next rule is checked.
3. If a messaging engine is running anywhere in the bus, then the application makes a remote connection to the selected messaging engine. If there is no suitable messaging engine, the connection attempt does not succeed.

For an application running outside an application server, connection requests are workload balanced across all the available messaging engines in the bus.

In both cases (that is, an application running in an application server and an application running outside an application server) you can restrict the range of messaging engines available for connection, to a subgroup of those available in the service integration bus. You do this by configuring the following connection properties of the connection factory or activation specification:

Target The name of a target that identifies a group of messaging engines. Specify the type of target using the Target type property.

Before the connection proximity search is performed to select a suitable messaging engine, the set of messaging engines that are members of the specified target group are selected. The connection proximity search is then restricted to these messaging engines. If a target group is not specified (the default), then all messaging engines in the bus are considered during the connection proximity search.

For example, if the Target type property is set to Bus member name, the Target property specifies the name of the bus member from which suitable messaging engines can be chosen.

Target type
The type of target named in the Target property.

Bus member name
The name of a bus member. This option retrieves the active messaging engines that are hosted by the named bus member (an application server or server cluster).

Custom messaging engine group name
The name of a custom group of messaging engines (that form a self-declaring cluster). This option retrieves the active messaging engines that have registered with the named custom group.

Messaging engine name
The name of a messaging engine. This option retrieves the available endpoints that can be used to reach the named messaging engine.

Target significance

This property defines whether the connection proximity search is restricted to only the messaging engines in the target group.

Preferred

It is preferred that a messaging engine is selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine is not available in the target group, a messaging engine outside the target group is selected if available in the same service integration bus.

Required

It is required that a messaging engine is selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine is not available in the target group, the connection process fails.

Target inbound transport chain

The name of the messaging engine inbound transport chain that the application should target when connecting to a messaging engine in a separate process to the application.

These transport chains specify the communication protocols that can be used to communicate with the application server to which the client application is connected. If a messaging engine in another process is chosen, a connection can be made only if the messaging engine is in a server that runs the specified inbound transport chain.

The following predefined messaging engine inbound transport chains are provided:

InboundBasicMessaging

JFAP over TCP/IP

InboundSecureMessaging

JFAP over SSL over TCP/IP

Connection proximity

For an application running in an application server, this property defines the proximity of messaging engines relative to the application server. For an application running outside an application server, this property defines the proximity of messaging engines relative to the bootstrap server.

Bus Connections can be made to messaging engines in the same bus.

A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host, and in turn ahead of a suitable messaging engine in another host.

Cluster

Connections can be made to messaging engines in the same server cluster. If the application is not running in a clustered server, or the bootstrap server is not in a cluster, then there are no suitable messaging engines.

A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host, and in turn ahead of a suitable messaging engine in another host.

This option should be used only in WebSphere Application Server environments that support server clusters.

Host Connections can be made to messaging engines in the same host. A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host.

Server

Connections can be made to messaging engines in the same application server.

To create or modify a JMS connection factory, see the following topics:

- Configuring the messaging engine selection process for JMS applications.
- Configuring a unified connection factory for the default messaging provider.
- createSIBJMSConnectionFactory command.
- modifySIBJMSConnectionFactory command.

To create or modify an activation specification, see the following topics:

- Configuring an activation specification for the default messaging provider.
- createSIBJMSActivationSpec command.
- modifySIBJMSActivationSpec command.

Why and when to pass the JMS message payload by reference

When large object messages or bytes messages are sent, the cost in memory and processor use of serializing, deserializing, and copying the message payload can be significant. If you enable the **`pass message payload by reference`** properties on a connection factory or activation specification, you tell the default messaging provider to override the JMS 1.1 specification and potentially reduce or bypass this data copying.

Background

The JMS 1.1 specification states that object messages are passed by value. This means that a JMS provider such as the default messaging provider in WebSphere Application Server has to take a copy of the object in `ObjectMessage` at the time the object is set into the message payload, in case the client application modifies the object after setting it. In practice this means serializing it, as there is no other entirely safe way to take a copy. The specification also states that when a consumer application gets the data from the message, the JMS provider must create and return a copy of that data.

If you enable the “pass message payload by reference” properties, you might get memory and performance improvements for JMS messaging.

CAUTION:

- **The parts of the JMS specification that are bypassed by these properties are defined to ensure message data integrity.**
- **Any of your JMS applications that use these properties must strictly follow the rules that are described in detail later in this section, or you risk losing data integrity.**
- **You should read and understand this entire topic before enabling these properties.**

To pass the message payload by reference, you set the following properties on connection factories and activation specifications:

`producerDoesNotModifyPayloadAfterSet` (for connection factories) or `forwarderDoesNotModifyPayloadAfterSet` (for activation specifications)

When this property is enabled, object or bytes messages produced by the connection factory or forwarded through the activation specification are not copied when set into the message and are only serialized when absolutely necessary. Applications sending such messages must not modify the data after it has been set into the message.

`consumerDoesNotModifyPayloadAfterGet`

When this property is enabled, object messages received through the connection factory or activation specification are only serialized when absolutely necessary. The data obtained from those messages must not be modified by applications.

Potential benefits of passing the message payload by reference

The following table shows the conditions under which you might get performance benefits by enabling the “pass message payload by reference” properties. This table makes the following assumptions:

- Your JMS applications conform to the rules described in the next section of this topic.
- Your message producer and consumer applications run in the same JVM (server), along with the messaging engine that hosts the destination used by these applications.

Note:

- If your applications run in different servers, or on the z/OS platform (where WebSphere Application Server runs in multiple JVMs), then object messages are serialized and no performance benefits are gained for these messages. Bytes message benefits might still be gained.
- There are many internal runtime conditions that can cause your messages to be serialized, so even if your configuration meets all the conditions described in this topic you might gain little or no performance benefit from enabling the “pass message payload by reference” properties.

Table 27. How configuration and runtime factors determine what data is copied, when it is copied, and the potential performance benefit.. The first column lists the degree of potential performance benefits. The second column includes the configuration and runtime events of the potential benefits. The third column provides information such as what data is copied and when the data is copied based on the configuration and runtime events of the potential benefits.

Degree of potential performance benefit	Configuration and runtime events	When the data is copied
No potential benefit	The “pass message payload by reference” properties are not enabled (default behavior).	Object message data is copied as soon as it is set into the message and when it is retrieved from the message. Bytes message data is copied as soon as it is set into the message and when it is retrieved from the message.
Some potential benefit	The “pass message payload by reference” properties are enabled, and either or both of the following conditions are true: • The send or receive message is transacted. • The consumer is not available when the message is produced.	IBM i Object message data is only copied when necessary. Bytes message data is only copied when necessary.
Maximum potential benefit	The “pass message payload by reference” properties are enabled, and both of the following conditions are true: • Neither the send nor the receive message is transacted. • The consumer is waiting for the message when it is produced (for example if the consumer is a message-driven bean).	IBM i Object message data might never be copied. Bytes message data is only copied when necessary.

Rules that your JMS applications must obey

The parts of the JMS specification that are bypassed by the “pass message payload by reference” properties are defined to ensure message data integrity. If your JMS applications obey the rules given in the following table, then you can safely enable the “pass message payload by reference” properties on the connection factories and activation specifications that the applications use.

If you enable the “pass message payload by reference” properties for JMS applications that do not follow these rules, then the applications might receive exceptions or, more importantly, the integrity of the message data might be compromised.

Table 28. Rules that your JMS applications must obey, by application type. The first column lists the JMS application types. The second column provides the rules that the JMS application must follow.

Application type	**Rules**
JMS producer application	A JMS producer application that sends object messages must not alter the object after it is set into the payload of the message. A JMS producer application that sends bytes messages: • must write data into the message with a single call to `writeBytes(byte[])`. • must not alter the byte array after it is written into the message.
JMS consumer application	A JMS consumer application that receives object messages must not alter the payload it gets from the message.
JMS forwarder application **Note:** A JMS forwarder application receives a message (through a connection factory, or if it is a message-driven bean through an activation specification), then sends the message object on to another destination.	A JMS forwarder application that replaces the payload of the received message with a new payload: • (for object messages) must not alter the object after it is set into the payload of the message. • (for bytes messages): – must write data into the message with a single call to `writeBytes(byte[])`. – must not alter the byte array after it is written into the message.

Ensuring that your object messages can be serialized

Under normal JMS messaging conditions (that is, when the “pass message payload by reference” properties are not enabled), the data in an object message is serialized as soon as the object is passed to the messaging system, for example on `set` or `send`. If the message payload cannot be serialized, then an exception message is immediately returned to the client application.

When the “pass message payload by reference” properties are enabled, the message payload is accepted from the client application without attempting to serialize it. If the system later discovers that the data cannot be serialized, the system can no longer inform the client application that sent the message - and because the data is not serializable, the system cannot persist or transmit the complete message. The message and its properties are stored, but the user data inside the message (the payload) cannot be stored and is discarded. If there are serialization problems when the system tries to convert an object message into a data graph for a mediation, the message payload is discarded and the mediation receives a message with the data value set to null.

If your data cannot be serialized, then it is lost. Therefore you should first test your configuration without enabling the “pass message payload by reference” properties, to check that all data sent into the system is serializable.

When the system discovers that an object message cannot be serialized, it writes the following error message (JMS_IBM_ExceptionMessage) to the `SystemOut.log` file, where “{0}” is replaced by the class name of the failing object:

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

CWSIK0200E: An object of class "{0}" has been set into the message payload but it cannot be serialized.

Explanation: An object message sent with the `producerDoesNotModifyPayloadAfterSet` flag enabled on its connection factory was sent with a payload that was not serializable by the system. This message data has been lost.

Action: Disable the `producerDoesNotModifyPayloadAfterSet` on the connection factory. Without the flag enabled, the JMS application that sets the object into the message will receive any serialization exception immediately.

The following exception properties are used to indicate that a data object cannot be serialized and has been discarded. JMS applications can find out what has happened from the `JMS_IBM_Exception` properties. Mediations can find out what has happened from the `JMS_IBM_Exception` and `SI_Exception` properties.

JMS_IBM_ExceptionReason
SIRCConstants.SIRC0200_OBJECT_FAILED_TO_SERIALIZE

JMS_IBM_ExceptionTimestamp
The time at which the object failed to serialize, in `System.currentTimeMillis()` form.

JMS_IBM_ExceptionMessage
Message CWSIK0200E, as previously described.

SI_ExceptionReason
SIRC0200_OBJECT_FAILED_TO_SERIALIZE

SI_ExceptionTimestamp
The time at which the object failed to serialize, in `System.currentTimeMillis()` form.

SI_ExceptionInserts
A string array containing one entry. The entry contains the class name of the object.

Note: The most likely explanation as to why your data objects cannot be serialized is that you have written your own `writeObject()` or `writeExternal()` methods and have not fully tested every option (for example `NullPointer` exceptions, or `ArrayIndexOutOfBounds` exceptions).

Pass message payload by reference: Potential benefits for each processing step:

For each processing step taken by your JMS messaging application, check this table to see when and why there is a potential performance benefit in enabling the "pass message payload by reference" properties on the associated connection factory or activation specification.

When large object messages or bytes messages are sent, the cost in memory and processor use of serializing, deserializing, and copying the message payload can be significant. If you enable the **`pass message payload by reference`** properties on a connection factory or activation specification, you tell the default messaging provider to override the JMS 1.1 specification and potentially reduce or bypass this data copying.

CAUTION:
The parts of the JMS Specification that are bypassed by these properties are defined to ensure message data integrity. Any of your JMS applications that use these properties must strictly follow the rules that are described in the topic Why and when to pass the JMS message payload by reference, or you risk losing data integrity.

To pass the message payload by reference, you set the following properties on connection factories and activation specifications:

producerDoesNotModifyPayloadAfterSet (for connection factories) or forwarderDoesNotModifyPayloadAfterSet (for activation specifications)

When this property is enabled, object or bytes messages produced by the connection factory or forwarded through the activation specification are not copied when set into the message and are only serialized when absolutely necessary. Applications sending such messages must not modify the data after it has been set into the message.

consumerDoesNotModifyPayloadAfterGet

When this property is enabled, object messages received through the connection factory or activation specification are only serialized when absolutely necessary. The data obtained from those messages must not be modified by applications.

Table 29. Potential performance benefits for each processing step taken by your producer, consumer or forwarder application. The first column of the table lists the processing steps for the objects and bytes messages. The second column indicates if there is a possibility for performance improvement. The third column provides a brief explanation about the processes and the properties.

Processing step	Is there potential for performance improvement?	Explanation
Object messages with producer and consumer applications		
An object message is sent to a consumer application in the same JVM. The producerDoesNotModifyPayloadAfterSet and consumerDoesNotModifyPayloadAfterGet properties are both enabled.	Yes	Under certain conditions, the payload object is passed by reference to the consumer application. If the message is not persistent or transacted, and the consumer application is immediately available, the payload object might never be serialized.
An object message is produced with the producerDoesNotModifyPayloadAfterSet property enabled, then received by a consumer application for which the consumerDoesNotModifyPayloadAfterGet property is not enabled.	No	No benefit is gained because the consumer application does not have the consumerDoesNotModifyPayloadAfterGet property enabled.
An object message is sent to a single consumer application in a different JVM. The producerDoesNotModifyPayloadAfterSet and consumerDoesNotModifyPayloadAfterGet properties are both enabled.	No	No benefit is gained because the single consumer application is in another JVM.
An object message is received by multiple consumer applications (a topic) that all have the consumerDoesNotModifyPayloadAfterGet property enabled and are all running in the same JVM.	Yes	All consumer applications with the consumerDoesNotModifyPayloadAfterGet property enabled might receive a reference to the same object, though this is not guaranteed.
Object messages with forwarder applications		
An object message is forwarded and the forwarder application accesses the payload of the message. The consumerDoesNotModifyPayloadAfterGet and producer/forwarderDoesNotModifyPayloadAfterSet properties are enabled.	Yes	There is a potential performance benefit in the following cases: • The producer of the forwarded message has the producerDoesNotModifyPayloadAfterSet property enabled and is in the same JVM as the forwarder application. • The consumer of the forwarded message has the consumerDoesNotModifyPayloadAfterGet property enabled and is in the same JVM as the forwarder application.
An object message is forwarded and the forwarder application accesses the payload of the message. Only the producer/ forwarderDoesNotModifyPayloadAfterSet property is enabled.	Yes	There is a potential performance benefit if the consumer of the forwarded message has the consumerDoesNotModifyPayloadAfterGet property enabled and is in the same JVM as the forwarder application.

Table 29. Potential performance benefits for each processing step taken by your producer, consumer or forwarder application (continued). The first column of the table lists the processing steps for the objects and bytes messages. The second column indicates if there is a possibility for performance improvement. The third column provides a brief explanation about the processes and the properties.

Processing step	Is there potential for performance improvement?	Explanation
An object message is forwarded and the forwarder application does not access the payload of the message.	No	If the payload of the object message is not accessed by the forwarder application then the producer/forwarderDoesNotModifyPayloadAfterSet and consumerDoesNotModifyPayloadAfterGet properties have no effect for the forwarder application. Benefits gained from the original producer application and ultimate consumer application are maintained.
Object messages with mediations		
An object message is sent to a mediated service integration bus destination.	No	If the object message is sent to a mediated destination then no performance benefit is gained by enabling the producerDoesNotModifyPayloadAfterSet property.
Bytes messages with consumer applications		
A bytes message is sent to any consumer application. The producerDoesNotModifyPayloadAfterSet property is enabled.	Yes	The part of the JMS specification that mandates copying the data on setting into the message is bypassed, saving a copy of the bytes data.
A bytes message is received with the consumerDoesNotModifyPayloadAfterGet property enabled.	No	The JMS API does not allow bytes data to be passed by reference back to consumer code (BytesMessage.readBytes methods).
A bytes message is received by multiple consumers (a topic) that all have the consumerDoesNotModifyPayloadAfterGet property enabled.	No	The JMS API does not allow bytes data to be passed by reference back to consumer code (BytesMessage.readBytes methods).
Bytes messages with forwarder applications		
A bytes message is forwarded.	No	There is no benefit gained from enabling any of the “pass message payload by reference” properties.

Pass message payload by reference: Example code for producer and consumer applications:

Code your JMS applications so that you can safely pass message payloads by reference for asynchronous messaging between producer and consumer applications within a single server.

When large object messages or bytes messages are sent, the cost in memory and processor use of serializing, deserializing, and copying the message payload can be significant. If the producer and consumer applications are in the same JVM and you enable the `pass message payload by reference` properties on the associated connection factories and activation specifications, message payloads can be passed by reference from producer application to consumer application. This can reduce or bypass the data copying and improve performance and memory use.

In the following figure, messages pass from a JMS producer application, through a producer connection factory, to a queue on a messaging engine. They are then taken off the queue and passed through a consumer connection factory or activation specification, to a JMS consumer application.

CAUTION:
The parts of the JMS Specification that are bypassed by these properties are defined to ensure message data integrity. Any of your JMS applications that use these properties must strictly follow the rules that are described below, or you risk losing data integrity.

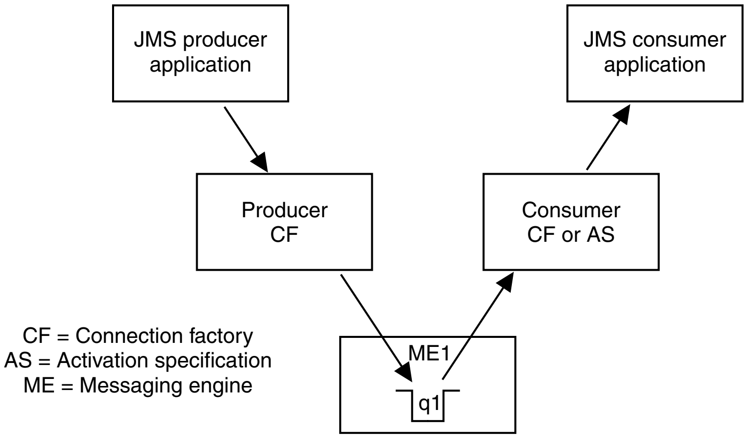


Figure 26. Producing and consuming messages

If you enable the **producerDoesNotModifyPayloadAfterSet** property for the producer connection factory, your producer application must guarantee not to modify the payload object after it has been set into object or bytes messages. To help you achieve this, here is some example code that you can adapt for use in your application:

```
DataObject data = new DataObject();
data.setXXX("xxx");
data.setYYY(yyy);
ObjectMessage message = session.createObjectMessage();
message.setObject(data);
data = null;
producer.send(message);
```

For bytes messages, your producer application must also guarantee to write only a single full byte array into the message. To help you achieve this, here is some example code that you can adapt for use in your application:

```
byte [] data = myByteData;
BytesMessage message = session.createBytesMessage();
message.writeBytes(data);
data = null;
producer.send(message);
```

If you enable the **consumerDoesNotModifyPayloadAfterGet** property for the consumer connection factory or activation specification, your consumer application must guarantee not to modify the payload it gets from the object message (consumption of bytes messages is not affected by the **consumerDoesNotModifyPayloadAfterGet** property). To help you achieve this, here is some example code that you can adapt for use in your application:

```
public void onMessage (Message message)
{
   ObjectMessage oMessage = (ObjectMessage) message;
   DataObject data = oMessage.getObject();
   System.out.print(data.getXXX());
   System.out.print(data.getYYY());
}
```

Pass message payload by reference: Usage scenarios and example code for forwarding applications:

A JMS forwarder application receives a message (through a connection factory, or if it is a message-driven bean through an activation specification), then sends the message object on to another destination.

Explore the different usage scenarios, then code your JMS forwarding applications so that you can safely pass message payloads by reference when forwarding messages from one queue to another within a single server.

When large object messages or bytes messages are sent, the cost in memory and processor use of serializing, deserializing, and copying the message payload can be significant. If you enable the **`pass message payload by reference`** properties on a connection factory or activation specification, you tell the default messaging provider to override the JMS 1.1 specification and potentially reduce or bypass this data copying.

In the following figure, messages pass from queue1 on a messaging engine, through a consumer activation specification or connection factory, to a JMS forwarding application. They are then forwarded through a producer connection factory to queue2 on the same messaging engine.

CAUTION:
The parts of the JMS Specification that are bypassed by these properties are defined to ensure message data integrity. Any of your JMS applications that use these properties must strictly follow the rules that are described below, or you risk losing data integrity.

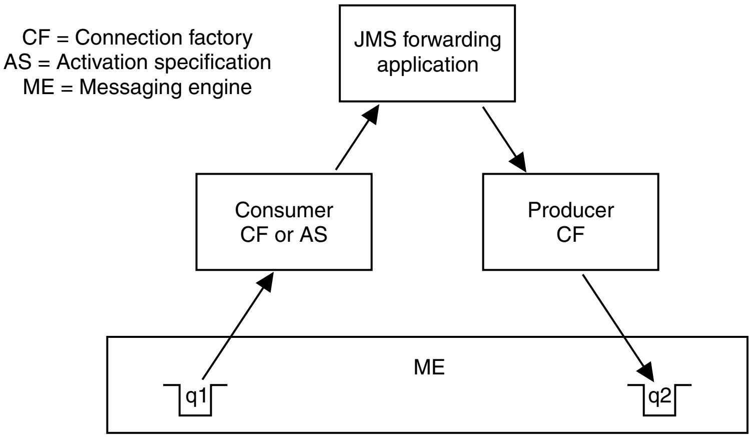


Figure 27. Forwarding messages

To understand the usage scenarios and associated example code given in this topic, you must note these important characteristics of a JMS forwarding application:

- A forwarding application does not replace the message object. This is useful if your application is just logging or otherwise recording (for example, printing out) the message before forwarding it, and also means that the forwarded message retains some useful message properties such as the `JMSCorrelationID`, `JMSReplyTo` and `JMSType` properties.
- A forwarding application can modify or replace the message payload. If it replaces the payload, it sets the new payload in the message object and changes the payload reference to point to the new message payload.
- For a forwarding application, the forwarded message is “created” and configured by the consumer connection factory or activation specification. The producer connection factory is used solely to route the forwarded message and has no effect upon the contents of the forwarded message.

The following table describes the four forwarding application usage scenarios that affect how you set the “pass message payload by reference” properties. Note that, because the producer connection factory has no effect upon the contents of the forwarded message, you set both the consumer properties and the producer/forwarder properties on the consumer connection factory or activation specification.

Table 30. Effect of the “pass message payload by reference” property settings on the forwarding application usage scenarios. The first column of the table lists the four forwarding application usage scenarios. The second column indicates the consumer property setting for the scenarios. The third column indicates the connection or the activation specification property setting for the scenarios.

Forwarding application usage scenario	**consumerDoesNotModify PayloadAfterGet** **property setting**	**producerDoesNotModify PayloadAfterSet** **(for connection factories) or** **forwarderDoesNotModify PayloadAfterSet** **(for activation specifications) property setting**
Scenario 1: The application receives a message, looks at the payload but does not modify it, and forwards the message on without modifying or replacing the payload.	Enabled	Not required, but can be enabled
Scenario 2: The application receives a message, looks at the payload but does not modify it, replaces the payload in the message with a new payload and forwards the message on without modifying the payload after the call to set it into the message.	Enabled	Enabled
Scenario 3: The application receives a message, looks at and modifies the payload, then sets the modified payload or some other data back into the message and forwards the message on without further modifying the payload after the call to set it into the message.	NOT enabled	Enabled
Scenario 4: The application receives a message, looks at and modifies the payload, then sets the modified payload or some other data back into the message, then further modifies the payload after the call to set it into the message.	NOT enabled	NOT enabled

For scenarios 1, 2 and 3 you can enable one or more of the “pass message payload by reference” properties, provided that your forwarding application can guarantee to behave as described in the scenario. To help you achieve this, here is some example code that you can adapt for use in your applications.

Forwarding application: scenario 1

The application receives a message, looks at the payload but does not modify it, and forwards the message on without modifying or replacing the payload.

```
public void onMessage (Message message)
{
   ObjectMessage oMessage = (ObjectMessage) message;
   DataObject data = oMessage.getObject();
   System.out.print(data.getXXX());
   System.out.print(data.getYYY());

   // get a session to forward on the received message

   producer.send(message);
   session.close();
}
```

Forwarding application: scenario 2

The application receives a message, looks at the payload but does not modify it, replaces the payload in the message with a new payload and forwards the message on without modifying the payload after the call to set it into the message.

```
public void onMessage (Message message)
{
   ObjectMessage oMessage = (ObjectMessage) message;
   DataObject data = oMessage.getObject();
   System.out.print(data.getXXX());
   System.out.print(data.getYYY());

   // get a session to forward on the received message

   message.setObject(newData);

   producer.send(message);
   session.close();
}
```

For bytes messages, your application must also guarantee to write only a single full byte array into the message.

```
byte [] data = myByteData;
BytesMessage message = session.createBytesMessage();
message.writeBytes(data);
data = null;
producer.send(message);
```

Forwarding application: scenario 3

The application receives a message, looks at and modifies the payload, then sets the modified payload or some other data back into the message and forwards the message on without further modifying the payload after the call to set it into the message.

```
public void onMessage (Message message)
{
   ObjectMessage oMessage = (ObjectMessage) message;
   DataObject data = oMessage.getObject();
   System.out.print(data.getXXX());
   System.out.print(data.getYYY());

   // get a session to forward on the received message

   data.setXXX(xxx);
   data.setYYY(yyy);
   message.setObject(data);

   producer.send(message);
   session.close();
}
```

For bytes messages, your application must also guarantee to write only a single full byte array into the message.

```
byte [] data = myByteData;
BytesMessage message = session.createBytesMessage();
message.writeBytes(data);
data = null;
producer.send(message);
```

Interoperation with WebSphere MQ

You can enable JMS interaction with a WebSphere MQ network by using the WebSphere MQ messaging provider. Service integration can also provide interoperation through a WebSphere MQ link or a WebSphere MQ server. Each type of connectivity is designed for different situations, and provides different advantages.

- For a comparison of the different ways of interoperating, see “Interoperation with WebSphere MQ: Comparison of architectures” on page 374 and “Interoperation with WebSphere MQ: Comparison of key features” on page 376.
- If you are not familiar with WebSphere MQ concepts, see “Interoperation with WebSphere MQ: Key WebSphere MQ concepts” on page 380.
- To understand about using WebSphere MQ as an external JMS messaging provider, see “Interoperation using the WebSphere MQ messaging provider” on page 382.
- To understand how best to develop your applications for interoperating with a WebSphere MQ network, see “How messages are passed between service integration and a WebSphere MQ network” on page 404.
- To understand the WebSphere MQ link solution, see “Interoperation using a WebSphere MQ link” on page 409.
- To understand the WebSphere MQ server solution, see “Interoperation using a WebSphere MQ server” on page 432.

For more information about WebSphere MQ, see the WebSphere MQ library.

When a WebSphere Application Server process or an application client process starts, and while this process is running, an amount of processing is performed to allow it to support WebSphere MQ-related functionality such as the WebSphere MQ messaging provider. By default this processing is performed regardless of whether any WebSphere MQ-related functionality is ever used. If you do not need to take advantage of any WebSphere MQ functionality, it is possible to disable all WebSphere MQ functionality in an application server or client process to give increased performance. For more information, see Disabling WebSphere MQ functionality in WebSphere Application Server.

Comparison of WebSphere Application Server and WebSphere MQ messaging

If you are not already an established user of either WebSphere Application Server or WebSphere MQ, and you are considering whether the service integration platform or WebSphere MQ better meets your messaging needs, use this table to compare the main features of the two platforms.

Table 31. Comparison of service integration and WebSphere MQ main features. The first column of this table lists the main features of service integration (the default messaging provider for WebSphere Application Server), and the second column lists the features of WebSphere MQ).

Service integration (the default messaging provider for WebSphere Application Server)	WebSphere MQ
Closely integrated with WebSphere Application Server, and combines well with the Java Platform, Enterprise Edition (Java EE)	Can connect to almost any platform, and supports a heterogeneous environment
Supports multiple languages through XMS clients, and multiple platforms	Supports multiple languages and multiple platforms
Limited tooling support, other than what is provided in WebSphere Application Server	Has many Independent Software Vendor (ISV) tools
Provides strong performance for both persistent and non-persistent messages for JMS	Supports JMS and non-JMS messaging interfaces, and provides strong performance for non-JMS applications
Designed for a maximum message size of about 40 megabytes on a 32-bit operating system (subject to heap usage)	Supports large message sizes up to about 100 megabytes

Table 31. Comparison of service integration and WebSphere MQ main features (continued). The first column of this table lists the main features of service integration (the default messaging provider for WebSphere Application Server), and the second column lists the features of WebSphere MQ).

Service integration (the default messaging provider for WebSphere Application Server)	WebSphere MQ
Underpins WebSphere Enterprise Service Bus and WebSphere Process Server	Underpins WebSphere MQ and WebSphere MQ File Transfer Edition
Included in a single administrative model for WebSphere Application Server, WebSphere Enterprise Service Bus, and WebSphere Process Server	Can integrate existing infrastructure and applications (for example, CICS)
Clustering is integrated with WebSphere Application Server clustering for high availability and scalability	WebSphere MQ clustering provides selective parallelism of clustered queues

Note: If your existing or planned messaging environment involves both WebSphere MQ and WebSphere Application Server systems, the messaging platform that you choose for a given task does not necessarily determine which JMS messaging provider you should use. For more information, see Choosing messaging providers for a mixed environment.

Interoperation with WebSphere MQ: Comparison of architectures

The three different ways that you can send messages between WebSphere Application Server and a WebSphere MQ network are compared at a high level, showing the relative advantages and disadvantages of each approach.

WebSphere MQ as an external messaging provider

The WebSphere MQ messaging provider does not use service integration. It provides JMS messaging access to WebSphere MQ from WebSphere Application Server.The WebSphere MQ messaging provider makes point-to-point messaging and publish/subscribe messaging available to WebSphere Application Server applications using the existing capabilities in the WebSphere MQ environment. WebSphere Application Server applications can interact with WebSphere MQ queues and topics to send, receive, publish, and subscribe to messages in the same way as any JMS application in the WebSphere MQ environment.

Using WebSphere MQ as an external messaging provider requires more WebSphere MQ administration, less WebSphere Application Server administration.

Table 32. Advantages and disadvantages of WebSphere MQ as an external messaging provider. The first column of this table shows the advantages of using WebSphere MQ as an external messaging provider, and the second column shows the disadvantages of using WebSphere MQ as an external messaging provider.

Advantages	Disadvantages
• You do not have to configure a service integration bus or messaging engines. • You can connect directly to WebSphere MQ queue managers. • You manage a single JMS messaging provider rather than two. • You can connect to queue managers in client mode or bindings mode. • You can use point-to-point messaging and publish/subscribe messaging.	• Interaction between WebSphere Application Server and WebSphere MQ is not seamless. • You cannot use service integration mediations for modifying messages, for routing, or for logging.

A WebSphere MQ network as a foreign bus (using WebSphere MQ links)

A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the

WebSphere MQ network.When you use a WebSphere MQ link, the messaging bus is seen by the WebSphere MQ network as a virtual queue manager, and the WebSphere MQ network is seen by service integration as a foreign bus. A WebSphere MQ link enables WebSphere Application Server applications to send point-to-point messages to WebSphere MQ queues (defined as destinations in the service integration bus), and allows WebSphere MQ applications to send point-to-point messages to destinations in the service integration bus (defined as remote queues in WebSphere MQ). You can also set up a publish/subscribe bridge so that WebSphere Application Server applications can subscribe to messages published by WebSphere MQ applications, and WebSphere MQ applications can subscribe to messages published by WebSphere Application Server applications. The link ensures that messages are converted between the formats used by WebSphere Application Server and those used by WebSphere MQ.

Using a WebSphere MQ network as a foreign bus (using WebSphere MQ links) requires more WebSphere Application Server administration, less WebSphere MQ administration.

Table 33. Advantages and disadvantages of a WebSphere MQ network as a foreign bus (using WebSphere MQ links). The first column of this table shows the advantages of using a WebSphere MQ network as a foreign bus (using WebSphere MQ links), and the second column shows the disadvantages of using a WebSphere MQ network as a foreign bus (using WebSphere MQ links).

Advantages	Disadvantages
• A WebSphere MQ client facility is not required on the gateway WebSphere MQ queue manager. • Each end of the link appears in natural form to the other; WebSphere MQ appears to service integration to be a (foreign) bus, service integration appears to WebSphere MQ to be a (virtual) queue manager. • Better performance over the link is possible when compared with WebSphere MQ servers or direct connection to WebSphere MQ as an external JMS messaging provider. • A managed connection from one node to another is created, but not from every application server in the cell. • You do not have to define individual WebSphere MQ queues to the service integration bus. • Good security support is provided. For example, you can control which users are allowed to put messages onto queues. • WebSphere Application Server and WebSphere MQ can exist on separate hosts. • Interaction between WebSphere Application Server and WebSphere MQ is seamless. • You can configure a publish/subscribe bridge, through which WebSphere Application Server applications can subscribe to messages published by WebSphere MQ applications, and WebSphere MQ applications can subscribe to messages published by WebSphere Application Server applications.	• You must configure a service integration bus and messaging engines. • You cannot connect to queue managers in bindings mode. • Optimum load balancing is less easy to achieve because messages have to be “pushed” from either end of the link. • You cannot use service integration mediations for modifying messages, routing, or logging.

A WebSphere MQ server (a queue manager or queue-sharing group) as a bus member

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. For interoperation with WebSphere Application Server Version 7 or later, the version of WebSphere MQ must be WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ (distributed platforms) Version 7 or later. A WebSphere MQ server supports the high availability and optimum load-balancing characteristics provided by a WebSphere MQ for z/OS network. A WebSphere MQ server defines the connection and quality of service properties used for the connection, and also ensures that messages are converted between the formats used by WebSphere Application Server and those used by WebSphere MQ. A WebSphere MQ server only represents queues for point-to-point messaging; it does not represent topics for publish/subscribe messaging.

Using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member requires more WebSphere Application Server administration, less WebSphere MQ administration.

Table 34. Advantages and disadvantages of a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member. The first column of this table shows the advantages of using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member, and the second column shows the disadvantages of using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member.

Advantages	Disadvantages
• WebSphere Application Server and WebSphere MQ can exist on separate hosts. • Each end of the connection appears in natural form to the other; WebSphere MQ queue manager appears to service integration to be a foreign bus, service integration appears to WebSphere MQ to be a client. • Close integration of applications is possible; service integration applications are able to consume messages directly from the WebSphere MQ network. • You can connect to queue managers in client mode or bindings mode. • You can use mediations for modifying messages, routing, or logging. • Good security support is provided. For example, you can control which users are allowed to put messages onto queues. • You can get messages from WebSphere MQ queues (GET). • Interaction between WebSphere Application Server and WebSphere MQ is seamless. • Queues on the WebSphere MQ network are automatically discovered.	• You must configure a service integration bus and messaging engines. • The queue managers and queue-sharing groups must be accessible from all the messaging engines in the bus. • You cannot use the WebSphere MQ server for publish/subscribe messaging with WebSphere MQ. • WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ (distributed platforms) Version 7 or later, is a prerequisite. • If you are using different nodes with WebSphere MQ for z/OS, depending on the number of nodes and your version of WebSphere MQ for z/OS, you might require the Client Attachment feature (CAF) on z/OS. • You must explicitly define all destinations.

Interoperation with WebSphere MQ: Comparison of key features

There are three different ways that you can send messages between WebSphere Application Server and a WebSphere MQ network. This topic compares the key features of each of the three ways.

Table 35. Key features comparison between the three ways of interoperating with WebSphere MQ. The first column of this table shows the key features of interoperating using the WebSphere MQ messaging provider with no bus, the second column shows the key features of interoperating using the WebSphere MQ network as a foreign bus (using WebSphere MQ links), and the third column shows the key features of interoperating using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member.

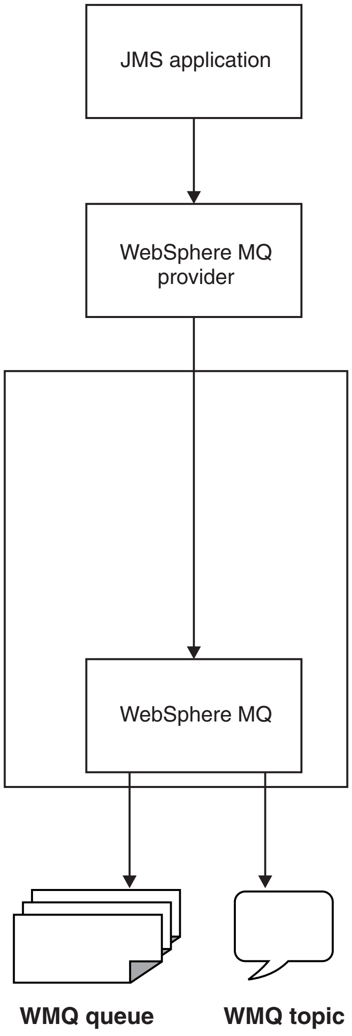
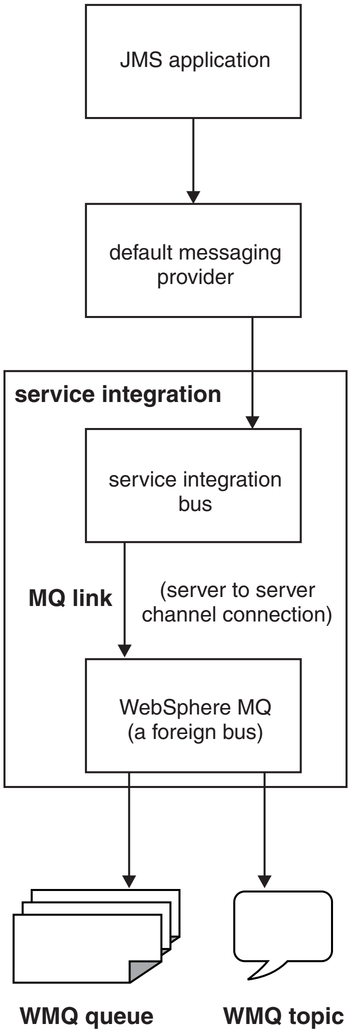
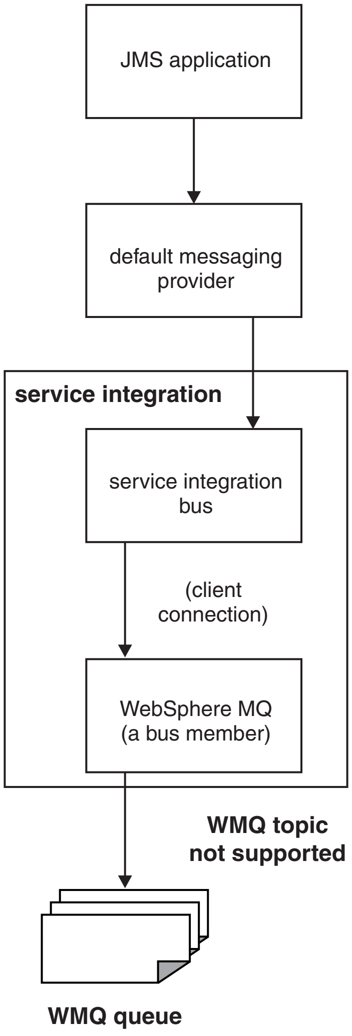
WebSphere MQ messaging provider (no bus)	**A WebSphere MQ network as a foreign bus (using WebSphere MQ links)**	**A WebSphere MQ server (a queue manager or queue-sharing group) as a bus member**
 JMS application WebSphere MQ provider WebSphere MQ WMQ queue WMQ topic	 JMS application default messaging provider service integration service integration bus MQ link (server to server channel connection) WebSphere MQ (a foreign bus) WMQ queue WMQ topic	 JMS application default messaging provider service integration service integration bus (client connection) WebSphere MQ (a bus member) WMQ topic not supported WMQ queue
Connectivity		
With the WebSphere MQ messaging provider, you can interoperate with WebSphere MQ Version 6 or later. See the following link, for details on which fix pack levels you will need for your configuration: http://www-01.ibm.com/support/docview.wss?uid=swg21498708	With a WebSphere MQ link, you can interoperate with any supported version or release of WebSphere MQ, on any platform.	With a WebSphere MQ server, you can only interoperate with WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ Version 7 or later.
Uses the WebSphere MQ messaging provider.	Uses the default messaging provider.	Uses the default messaging provider.
No use of service integration buses.	Uses a service integration bus.	Uses a service integration bus.

Table 35. Key features comparison between the three ways of interoperating with WebSphere MQ (continued). The first column of this table shows the key features of interoperating using the WebSphere MQ messaging provider with no bus, the second column shows the key features of interoperating using the WebSphere MQ network as a foreign bus (using WebSphere MQ links), and the third column shows the key features of interoperating using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member.

WebSphere MQ messaging provider (no bus)	A WebSphere MQ network as a foreign bus (using WebSphere MQ links)	A WebSphere MQ server (a queue manager or queue-sharing group) as a bus member
WebSphere Application Server regards the WebSphere MQ messaging provider as a JMS messaging provider. The WebSphere MQ messaging provider is regarded by the WebSphere MQ network as a WebSphere MQ client attaching to the queue manager or queue-sharing group.	Each end of the WebSphere MQ link appears in a natural form to the other end, so the WebSphere MQ network appears to service integration as a foreign bus and the service integration bus appears as a virtual queue manager to the WebSphere MQ network.	The WebSphere MQ server regards the WebSphere MQ queue manager or queue-sharing group as a bus member, or a mechanism for queuing messages for the service integration bus. A queue is viewed as a bus destination. The WebSphere MQ server is regarded by the WebSphere MQ network as a WebSphere MQ client attaching to the queue manager or queue-sharing group.
Provides multiple connections between WebSphere Application Server application servers and WebSphere MQ queue managers or queue-sharing groups. Connections are established as and when required to allow WebSphere Application Server applications to access WebSphere MQ queues.	Provides a single connection between a service integration bus and a WebSphere MQ network (comprising one or more interconnected WebSphere MQ queue managers or queue-sharing groups). This single connection is used to transfer all the messages that are exchanged between the service integration network and the WebSphere MQ network. The link acts as a funnel, routing messages through the gateway messaging engine or queue manager. If you want to establish multiple links from a service integration network, you can define multiple foreign buses to represent different queue managers or queue-sharing groups on the WebSphere MQ network.	Provides multiple connections between messaging engines in a service integration bus and WebSphere MQ queue managers or queue-sharing groups. Connections are established as and when required, to allow WebSphere Application Server applications to access WebSphere MQ queues. A connection can be configured to use properties of the message bus to which it belongs, giving the potential for each WebSphere MQ server to be bus-specific.
Connection between the WebSphere Application Server and the WebSphere MQ network can use a TCP/IP communication link or, if the WebSphere Application Server is running on the same image as the WebSphere MQ queue manager, it can use a direct call interface (this is called *bindings mode*). The channel for the connection is a bidirectional MQI channel.	Connection between the service integration bus network and the WebSphere MQ network uses a TCP/IP communication link. The sender and receiver channels for the connection are message channels.	Connection between the service integration bus network and the WebSphere MQ network can use a TCP/IP communication link or, if the WebSphere Application Server application server is running on the same image as the WebSphere MQ queue manager, it can use a direct call interface (this is called *bindings mode*). The channel for the connection is a bidirectional MQI channel.
For WebSphere MQ for z/OS, messages can be stored on shared queues. If a queue manager fails, messages can still be retrieved from a different queue manager (so no single point of failure exists).	If the communication link fails temporarily, messages are stored by WebSphere MQ or the service integration bus and are delivered when the communication link recovers.	For WebSphere MQ for z/OS, messages can be stored on shared queues. If a queue manager fails, messages can still be retrieved from a different queue manager (so no single point of failure exists).
Applications		
Does not integrate the service integration bus with the WebSphere MQ network. Service integration bus mediations running in WebSphere Application Server cannot process messages from a WebSphere MQ queue, and WebSphere MQ applications cannot use WebSphere MQ servers to put messages to, or get messages from, service integration bus queue-type destinations.	Integrates the service integration bus with the WebSphere MQ network through a gateway queue manager. Traffic can be indirect, routed to a mapped queue.	Allows closer integration; messaging applications can directly produce messages to, and consume messages from WebSphere MQ queues.

Table 35. Key features comparison between the three ways of interoperating with WebSphere MQ (continued). The first column of this table shows the key features of interoperating using the WebSphere MQ messaging provider with no bus, the second column shows the key features of interoperating using the WebSphere MQ network as a foreign bus (using WebSphere MQ links), and the third column shows the key features of interoperating using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member.

WebSphere MQ messaging provider (no bus)	**A WebSphere MQ network as a foreign bus (using WebSphere MQ links)**	**A WebSphere MQ server (a queue manager or queue-sharing group) as a bus member**
WebSphere Application Server applications can send messages to WebSphere MQ queues. Sent messages are immediately added to the queue. If the WebSphere MQ queue is unavailable, applications cannot send messages.	WebSphere Application Server applications can send messages to WebSphere MQ queues. Sent messages are stored by the service integration bus for transmission to WebSphere MQ (this is called *store and forward* messaging). Applications can continue to send messages if the WebSphere MQ queue is unavailable.	WebSphere Application Server applications can send messages to WebSphere MQ queues. Sent messages are immediately added to the queue. If the WebSphere MQqueue is unavailable, applications cannot send messages.
WebSphere Application Server applications can receive messages from WebSphere MQ queues. The applications can use message consumers to receive messages, and message-driven beans can be configured to process messages as soon as they arrive at the WebSphere MQ queue.	WebSphere Application Server applications cannot receive messages from WebSphere MQ queues, because the queues are destinations in a foreign bus. For messages to pass from WebSphere MQ to WebSphere Application Server applications, WebSphere MQ applications must send the messages to a suitable destination in the service integration bus used by the WebSphere Application Server applications.	WebSphere Application Server applications can receive messages from WebSphere MQ queues. The applications can use message consumers to receive messages, and message-driven beans can be configured to process messages as soon as they arrive at the WebSphere MQ queue. Also, service integration bus mediations running in WebSphere Application Server can process messages as they arrive at a WebSphere MQ queue.
WebSphere Application Server applications can publish messages to WebSphere MQ topics and subscribe to messages on WebSphere MQ topics in the same way as applications in the WebSphere MQ environment.	You can set up a publish/subscribe bridge on the WebSphere MQ link, so that WebSphere Application Server applications and WebSphere MQ applications can publish or subscribe to selected topics that exist in both the WebSphere MQ environment and the WebSphere Application Server environment.	A WebSphere MQ server provides connections with queues for point-to-point messaging. A topic for publish/subscribe messaging cannot be associated with a WebSphere MQ server.
Messages are stored on queues, not messaging engines; one or many WebSphere Application Server applications can access the messages, even when the applications are running on different servers.	Messages are stored on messaging engines.	Messages are stored on queues, not messaging engines; one or many WebSphere Application Server applications can access the messages, even when the applications are running on different servers.
Messages are pulled from the queue by a consuming application, and pushed by a producing application.	Messages are pushed across the link, regardless of whether a consumer is ready.	Messages are pulled from the queue by a WebSphere Application Server consumer, and pushed by a WebSphere Application Server producer.
Does not support mediations.	Does not support mediations.	Supports different mediation scenarios for modifying message content, or routing, and for logging.
Optimum load balancing is easier to achieve because applications can pull messages from the WebSphere MQ network.	Messages are pushed to applications from the WebSphere MQ network, but workload balancing options are available in WebSphere Application Server.	Optimum load balancing is easier to achieve because applications can pull messages from the WebSphere MQ network.
Administration and security		
Configured and managed by using the administrative console.	Configured and managed by using the administrative console.	Configured and managed by using the administrative console. Automatically discovers queues on the WebSphere MQ network during configuration and administration.

Table 35. Key features comparison between the three ways of interoperating with WebSphere MQ (continued). The first column of this table shows the key features of interoperating using the WebSphere MQ messaging provider with no bus, the second column shows the key features of interoperating using the WebSphere MQ network as a foreign bus (using WebSphere MQ links), and the third column shows the key features of interoperating using a WebSphere MQ server (a queue manager or queue-sharing group) as a bus member.

WebSphere MQ messaging provider (no bus)	A WebSphere MQ network as a foreign bus (using WebSphere MQ links)	A WebSphere MQ server (a queue manager or queue-sharing group) as a bus member
Administration is carried out in WebSphere MQ. In WebSphere Application Server you need to define JMS artefacts such as destinations, connection factories, listener ports, and activation specifications.	Cooperative administrative domains for WebSphere MQ and WebSphere Application Server: • Mutually agree definitions of channels, foreign destinations and buses, to reflect WebSphere MQ connectivity • Both ends of the link must be started • Administrators can stop or start a link	Independent administrative domains for WebSphere MQ and WebSphere Application Server: • Separate authority • Temporal decoupling of administrative changes
You might have to define server connection channels in WebSphere MQ.	You must define partner channel definitions in WebSphere MQ.	You might have to define server connection channels in WebSphere MQ.
Permission for WebSphere Application Server applications and mediations to send messages to, and receive messages from, a particular WebSphere MQ is controlled by WebSphere MQ administration.	Permission for WebSphere Application Server applications to send messages to a particular WebSphere MQ queue is controlled by service integration bus administration. Permission for WebSphere MQ applications to send messages to service integration destinations is controlled by WebSphere MQ administration.	Permission for WebSphere Application Server applications and mediations to send messages to, and receive messages from, a particular WebSphere MQ queue is controlled by service integration bus administration. Permission for WebSphere Application Server (which includes permission for its applications and mediations) to access WebSphere MQ queues is controlled by WebSphere MQ administration.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

Interoperation with WebSphere MQ: Key WebSphere MQ concepts

If you are not familiar with basic WebSphere MQ concepts, read about the objects in WebSphere MQ that are important for interoperation with WebSphere Application Server.

Queues and topics

A queue is a data structure used to store messages. Application programs can use JMS or WebSphere MQ API calls to put messages on WebSphere MQ queues. Other applications can get the messages from the queues.

A topic is the subject of the information that is published in a publish/subscribe message. Instead of putting a message on a specific queue, application programs can publish a message to a topic. Other applications obtain the messages by subscribing to the topic to receive all the messages published to that topic.

When an application puts a message on a queue, only one copy of the message exists. Even if more than one application can get messages from the queue, only one consumer can receive each message.

However, when an application publishes a message to a topic, any number of subscribers can receive a copy of the message.

Queue managers and queue-sharing groups

Each WebSphere MQ queue is owned by a queue manager. The queue manager is responsible for maintaining the queues it owns, and for placing all the messages it receives onto the appropriate queues. Application programs connect to a queue manager when they want to put messages on queues. Queue managers can also put messages on queues as part of their normal operation.

From WebSphere MQ Version 7, each topic in WebSphere MQ is also owned by a queue manager. The queue manager receives messages from publishers, and subscriptions from subscribers. The queue manager is responsible for routing the published messages to the subscribers that have registered an interest in the topic of the messages. In earlier versions of WebSphere MQ, publish/subscribe messaging is handled by a publish/subscribe broker, not by queue managers.

In WebSphere MQ for z/OS, you can set up shared queues that can be accessed by several queue managers in a sysplex. Messages that are put onto shared queues are stored in list structures in a zSeries® Coupling Facility, and large messages have their message data held in a shared DB2 table.

The queue managers that can access the same set of shared queues form a group called a queue-sharing group. Each member of the queue-sharing group connects to a DB2 system to access shared definitions for WebSphere MQ objects, including queues and channels. Any queue manager in the group can retrieve the messages held on a shared queue. An application that wants to access one of the shared queues can therefore connect to any of the queue managers within the queue-sharing group, so the application does not depend on the availability of a specific queue manager.

Local queues, remote queues, and clusters

In a WebSphere MQ network, intercommunication is achieved by sending messages from one queue manager or (for WebSphere MQ for z/OS) queue-sharing group to another.

WebSphere MQ application programs can put messages onto a local queue, which is a queue on the queue manager to which the application is connected. A queue manager has a definition for each queue that it owns. A queue manager can also have definitions for the queues that other queue managers own. From the perspective of the local queue manager to which the application is connected, these other queues are remote queues, and the queue managers that own them are remote queue managers.

As well as putting messages onto a local queue, WebSphere MQ application programs connected to a local queue manager can put messages targeted at remote queues. WebSphere MQ must then transmit the messages to the remote queue managers that own the remote queues. When messages are destined for a WebSphere MQ queue on a remote queue manager, the local queue manager holds them in a transmission queue until it is ready to forward them to the remote queue manager. A transmission queue is a special type of local queue on which messages are stored until they can be successfully transmitted and stored at the remote queue manager.

WebSphere MQ queue managers can be connected to form a cluster, using any of the communications protocols that are available on your WebSphere MQ platform. When you group queue managers in a cluster, the queues are still hosted by the queue managers (so they are not shared queues). However, by connecting into the cluster, queue managers can send a message to any other queue manager in the cluster, and make some or all of the queues that they host available to every other queue manager in the cluster as cluster queues. You do not have to set up explicit definitions on each queue manager for each remote queue and for the connection to each remote queue manager. Each queue manager in the cluster also uses a single cluster transmission queue to hold messages for any of the other queue managers, so you do not have to set up a transmission queue for each remote queue manager.

From WebSphere MQ Version 7, you can also connect together WebSphere MQ queue managers that own topics for publish/subscribe messaging. You can group queue managers that own topics into a publish/subscribe cluster, with links between all members, or into a publish/subscribe hierarchy, with parent and child relationships between the connected queue managers. Publications and subscriptions to topics can be shared between all the queue managers in the cluster or hierarchy.

Message channels

WebSphere MQ messages, whether they are put onto queues or published to topics, are transmitted between queue managers through message channels. A message channel is a one-way communication link between two queue managers. It can carry messages destined for any number of queues or topics that the remote queue manager hosts, or for any number of target queue managers.

You can define the following types of message channel in WebSphere MQ:

- Sender-receiver channel
- Requester-server channel
- Requester-sender channel
- Server-receiver channel
- Cluster-sender channels
- Cluster-receiver channels

For example, to define the type of message channel called a sender-receiver channel, you define a sender channel at the sending end, which could be the local queue manager. Then you use the same name to define a receiver channel at the receiver end, which could be the remote queue manager. A message channel is unidirectional, so if you want messages to flow in both directions, you must define a second message channel in the opposite direction between the queue managers.

For queue managers in a cluster, you do not have to define message channels between each pair of queue managers. Instead, you define two message channels to connect each queue manager into the cluster: one cluster-receiver channel for receiving messages, and one cluster-sender channel by which the queue manager introduces itself and learns about the cluster. The queue manager can then send a message to any other queue manager in the cluster.

Do not confuse message channels with MQI channels. The MQI is the Message Queue Interface in WebSphere MQ, which applications use to interact with queue managers. An MQI channel is a type of connection that is used by a WebSphere MQ client application to connect to a queue manager that is running on another system, and to issue MQI calls to the queue manager.

Interoperation using the WebSphere MQ messaging provider

Through the WebSphere MQ messaging provider in WebSphere Application Server, Java Message Service (JMS) messaging applications can use your WebSphere MQ system as an external provider of JMS messaging resources.

WebSphere MQ is characterized as follows:

- Messaging is handled by a network of queue managers, each running in its own set of processes and having its own administration.
- Features such as shared queues (on WebSphere MQ for z/OS) and WebSphere MQ clustering simplify administration and provide dynamic discovery.
- Many IBM and partner products support WebSphere MQ with (for example) monitoring and control, high availability and clustering.
- WebSphere MQ clients can run within WebSphere Application Server (JMS), or almost any other messaging environment by using a variety of APIs.

If your business uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominantly WebSphere MQ network, the WebSphere MQ messaging provider is a logical choice. However, there can be benefits in using another provider. If you are not sure which provider combination is best suited to your requirements, see Choosing messaging providers for a mixed environment.

The WebSphere MQ messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as “unified” or “common” interfaces). This enables applications to use the same interfaces for both point-to-point and publish/subscribe messaging, and also enables both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, this approach is considered good practice for new applications. The domain-specific interfaces are supported for backwards compatibility for applications developed to use domain-specific queue interfaces, as described in section 1.5 of the JMS 1.1 specification.

The WebSphere MQ messaging provider also supports the Java EE Connector Architecture (JCA) 1.5 activation specification mechanism for message-driven beans (MDBs) across all platforms supported by WebSphere Application Server.

You can use WebSphere Application Server to configure WebSphere MQ resources for applications (for example queue connection factories) and to manage messages and subscriptions associated with JMS destinations. You administer security through WebSphere MQ.

WebSphere Application Server Version 6.x contained a WebSphere MQ JMS client to facilitate interaction with WebSphere MQ. For WebSphere Application Server Version 7.0 and later this has been replaced with the WebSphere MQ resource adapter. The resource adapter is included with WebSphere Application Server and does not need to be installed separately. Service updates to the resource adapter are usually applied automatically through WebSphere Application Server fix packs. For more information about maintaining the resource adapter, see Maintaining the WebSphere MQ resource adapter.

You can use the following four WebSphere MQ connection properties to configure the WebSphere MQ resource adapter used by the WebSphere MQ messaging provider. These properties affect the connection pool that is used by activation specifications:

- maxConnections
- connectionConcurrency (Setting this property only affects WebSphere Application Server 7 nodes. The property has no effect for WebSphere Application Server Version 8 or later nodes.)
- reconnectionRetryCount
- reconnectionRetryInterval

To connect to multi-instance WebSphere MQ queue managers, you can provide host and port information in the form of a connection name list, which a connection factory or activation specification uses to connect to a multi-instance queue manager.

WebSphere MQ queue or topic destination properties allow you to specify:

- Whether an application processes the RFH version 2 header of a WebSphere MQ message as part of the JMS message body.
- The format of the JMSReplyTo field.
- Whether an application can read or write the values of MQMD fields from JMS messages that have been sent or received using the WebSphere MQ messaging provider.
- Which message context options are specified when sending messages to a destination.

For more information about using WebSphere MQ with WebSphere Application Server, see the white papers and IBM Redbooks publications provided by WebSphere MQ; for example, through the WebSphere MQ library web page.

Network topologies: Interoperating by using the WebSphere MQ messaging provider

There are several network topologies, clustered and not clustered, that allow WebSphere Application Server to interoperate with WebSphere MQ by using WebSphere MQ as an external JMS messaging provider. For providing high availability, some topologies are more suitable than others.

For completeness, this topic describes a wide range of topologies, including clustered and highly available topologies. Note that, for clustering and high availability, you need to use the network deployment or z/OS version of the product.

Note: In this topic “application server” refers to an application server that is running on WebSphere Application Server and "queue manager" refers to a queue manager that is running on WebSphere MQ.

The WebSphere Application Server high availability framework eliminates single points of failure and provides peer to peer failover for applications and processes running within WebSphere Application Server. This framework also allows integration of WebSphere Application Server into an environment that uses other high availability frameworks, such as High Availability Cluster Multi-Processing (HACMP™), in order to manage non-WebSphere Application Server resources.

The subsequent examples show the main network topologies for interoperating withWebSphere MQ using the WebSphere MQ messaging provider. Each of the four examples describes two network topologies, with varying locations for the application servers and queue managers.

- “Interoperation when WebSphere Application Server application server is not clustered and WebSphere MQ queue manager is not clustered”
  - The application server and the queue manager run on different hosts
  - The application server and the queue manager run on the same host
- “Interoperation when WebSphere Application Server application servers are clustered but WebSphere MQ queue manager is not clustered” on page 386
  - The queue manager runs on a different host from any of the application servers
  - The application servers run on several hosts, one of which hosts a queue manager
- “Interoperation when WebSphere Application Server application servers are clustered and WebSphere MQ queue managers are clustered” on page 388
  - The queue managers run on different hosts from the application servers
  - The queue manager runs on the same hosts as the application servers
- “Connecting WebSphere Application Server application servers to WebSphere MQ for z/OS with queue-sharing groups” on page 392
  - The application servers and the queue managers run in the same LPAR
  - The application servers and the queue managers run in different LPARs

Interoperation when WebSphere Application Server application server is not clustered and WebSphere MQ queue manager is not clustered:

Application servers running on WebSphere Application Server and queue managers running on WebSphere MQ can connect to each other when neither of them are clustered. However, this setup can be vulnerable to failure.

Note: In this topic “application server” refers to an application server that is running on WebSphere Application Server and “queue manager” refers to a queue manager that is running on WebSphere MQ.

There are two topology options:

- The application server and the queue manager run on different hosts

- The application server and the queue manager run on the same host

The application server and the queue manager run on different hosts

The WebSphere MQ transport type for the connection is specified as “client”. A client connection is used when the application server and queue manager are running on different hosts. This is a TCP/IP network connection that is used to communicate with the queue manager. A client connection is also known as “socket attach”.

The following figure shows an application server and a queue manager running on different hosts.

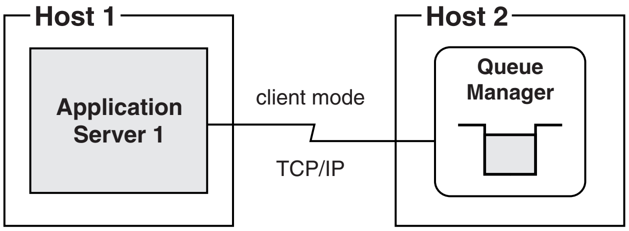


Figure 28. No clustering: client mode attachment to queue manager

This topology is vulnerable because inter-operation ceases if any of the following conditions occurs:
- The application server fails.
- The host on which the application server is running fails.
- The queue manager fails.
- The host on which the queue manager is running fails.

You can improve availability for this topology by using, for example, High Availability Cluster Multi-Processing (HACMP) to restart the failed component automatically.

The application server and the queue manager run on the same host

The transport type for the connection is specified as “bindings”. A bindings connection is used when the application server and the queue manager are running on the same host. This is a cross-memory connection that is used to communicate with a queue manager. A bindings connection is also known as “call attach”.

The following figure shows a application server and a queue manager running on the same host.

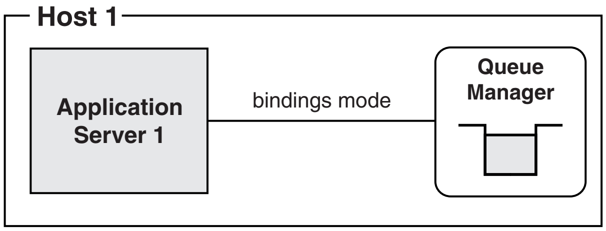


Figure 29. No clustering: bindings mode attachment to queue manager

The availability constraints for this topology are similar to the previous one. However, in some configurations bindings mode is faster and more processor efficient than client mode because the amount of processing is reduced.

Interoperation when WebSphere Application Server application servers are clustered but WebSphere MQ queue manager is not clustered:

Application servers running on WebSphere Application Server can be clustered together and connected to queue managers running onWebSphere MQ that are not clustered. This setup provides enhanced failover protection over non-clustered topologies.

Note: In this topic “application server” refers to an application server that is running on WebSphere Application Server and “queue manager” refers to a queue manager that is running on WebSphere MQ.

There are two topology options:

- The application servers run on several hosts, one of which hosts a queue manager
- The queue manager runs on a different host from any of the application servers

The queue manager runs on a different host from any of the application servers

In the subsequent figure:

- Application server 1, 2 and 3 are clustered in a WebSphere Application Server cluster.
- Application servers 1 and 3 are running on Host 1
- Application server 2 is running on Host 2
- Queue manager is running on Host 3
- A “client” connection is used when the application server and queue manager are running on different hosts. This is a TCP/IP network connection that is used to communicate with the queue manager. A client connection is also known as “socket attach”.
  - Application servers 1, 2 and 3 are connected to queue manager in client mode.

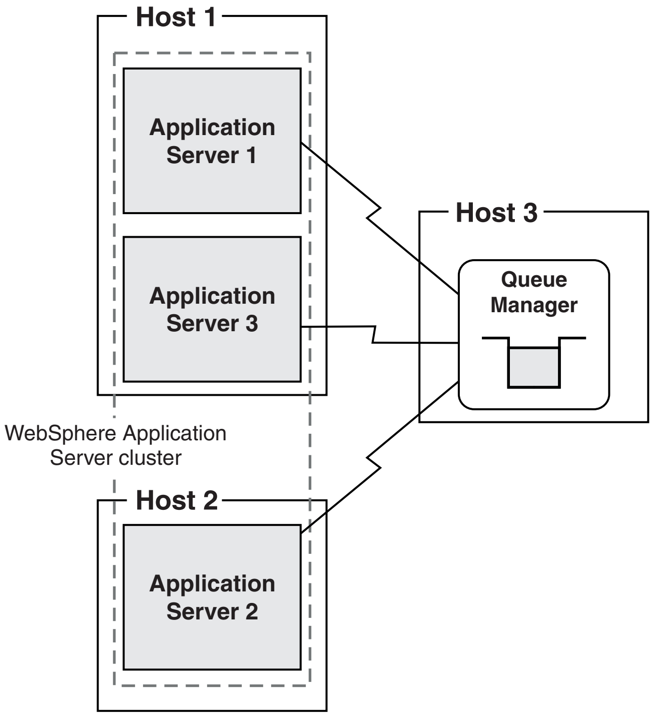


Figure 30. WebSphere Application Server clustering: client mode attachment to queue manager

- If any clustered application server fails, or the host on which it is running fails, the remaining application servers in the cluster can take over its workload.
- If the queue manager fails, or the host on which it is running fails, interoperation ceases.

You can improve availability for this topology by using, for example, High Availability Cluster Multi-Processing (HACMP) to restart the failed queue manager automatically.

The application servers run on several hosts, one of which hosts a queue manager

The following figure shows some application servers that are running on the same host as the queue manager. Other application servers in the same WebSphere Application Server cluster run on a different host.

In the subsequent figure:

- Application server 1, 2 and 3 are clustered in a WebSphere Application Server cluster.
- Application servers 1 and 3 are running on Host 1.
- Application server 2 is running on Host 2.
- Queue manager is running on Host 1.
- The transport type for the connection is specified as “bindings”. A “bindings” connection is used when the application server and the queue manager are running on the same host. This is a cross-memory connection that is used to communicate with a queue manager. A bindings connection is also known as “call attach”.
  - Application servers 1, and 3 are connected to queue manager in bindings mode.
- A “client” connection is used when the application server and queue manager are running on different hosts. This is a TCP/IP network connection that is used to communicate with the queue manager. A client connection is also known as “socket attach”.
  - Application server 2 is connected to queue manager in client mode.

Note: For application servers that are running on the same host as a queue manager, the WebSphere MQ transport type for the connection is specified as “bindings then client” mode, that is, if an attempt at a bindings mode connection to the queue manager fails, a client mode connection is made. For application servers that are not running on the same host as the queue manager, the application server automatically uses client mode.

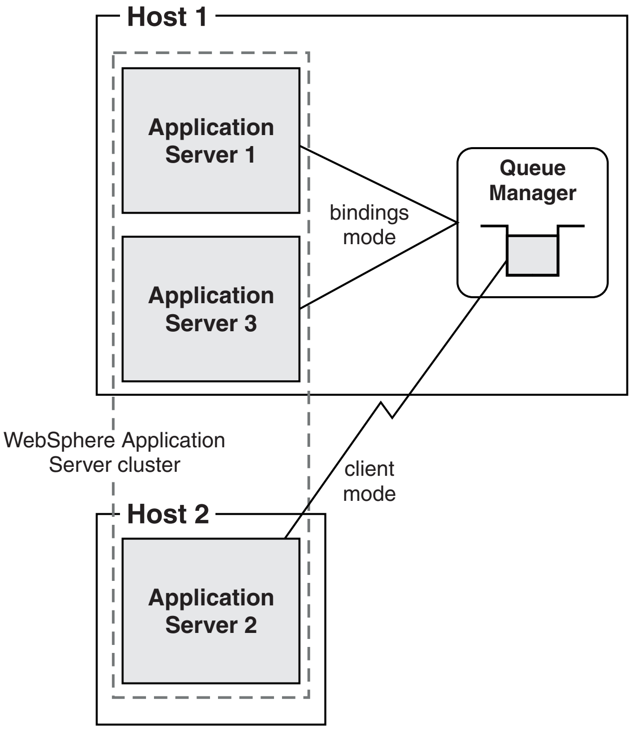


Figure 31. WebSphere Application Server clustering: bindings then client mode attachment to queue manager

- If one of the application servers fails, the remaining application servers in the cluster can take over its workload.
- If host 2 fails, application server 2 will stop. Application servers 1 and 3 can take over its workload.
- If the queue manager fails inter-operation ceases.
- If host 1 fails the queue manager, application server 1 and application server 3 will stop. Inter-operation will cease.

Interoperation when WebSphere Application Server application servers are clustered and WebSphere MQ queue managers are clustered:

WebSphere MQ queue managers are usually clustered in order to distribute the message workload and because, if one queue manager fails, the others can continue running.

Note: In this topic “application server” refers to an application server that is running on WebSphere Application Server and “queue manager” refers to a queue manager that is running on WebSphere MQ.

There are two topology options:

- The queue managers run on different hosts from the application servers
- The queue managers run on the same hosts as the application servers

The queue managers run on different hosts from the application servers

In the subsequent figure:

- Application server 1, 2 and 3 are clustered in a WebSphere Application Server cluster.
- Application servers 1 and 3 are running on Host 1.
- Application server 2 is running on Host 2.

- Queue managers 1, 2 and 3 are part of the same WebSphere MQ cluster.
- Queue manager 1 is running on Host 3.
- Queue manager 2 is running on Host 4.
- Queue manager 3 is running on Host 5.
- Queue manager 3 is responsible for distributing messages between the cluster queues in a way that achieves workload balancing.
- A "client" connection is used when the application server and queue manager are running on different hosts. This is a TCP/IP network connection that is used to communicate with the queue manager. A client connection is also known as “socket attach”.
  - Application servers 1 and 2 attach in client mode to queue manager 1.
  - Application server 3 attaches in client mode to queue manager 2.

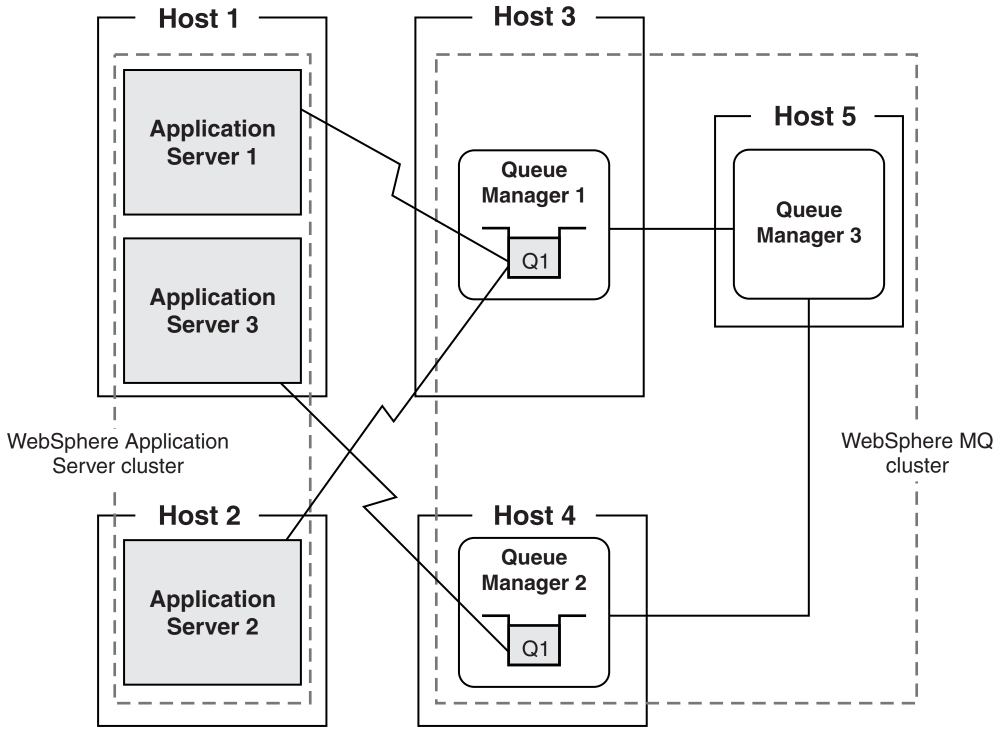


Figure 32. WebSphere Application Server clustering: client mode attachment to queue managers

If application server 1 fails:
Application server 2 can take over its workload because they are both attached to queue manager 1.

If application server 2 fails:
Application server 1 can take over its workload because they are both attached to queue manager 1.

If application server 3 fails:
You must restart it as soon as possible for the following reasons:

- Other application servers in the cluster can take over its external workload, but no other application server can take over its WebSphere MQ workload, because no other application server is attached to queue manager 2. The workload that was generated by application server 3 ceases.

- Queue manager 3 continues to distribute work between queue manager 1 and queue manager 2, even though the workload arriving at queue manager 2 cannot be processed by application server 1 or 2.

Note: If you choose not to restart, you can alleviate this situation by manually configuring Q1 on queue manager 2 so that the ability to put messages to it is inhibited. This results in all messages being sent to queue manager 1 where they are processed by the other application servers.

If queue manager 1 fails:

You should restart it as soon as possible for the following reasons:

- Messages that are on queue manager 1 when it fails are not processed until you restart queue manager 1.
- No new messages from WebSphere MQ applications are sent to queue manager 1, instead new messages are sent to queue manager 2 and consumed by application server 3.
- Because application servers 1 and 2 are not attached to queue manager 2, they cannot take on any of its workload.
- Because application servers 1, 2 and 3 are in the same WebSphere Application Server cluster, their non-WebSphere MQ workload continues to be distributed between them all, even though application servers 1 and 2 cannot use WebSphere MQ because queue manager 1 has failed.

Although this networking topology can provide availability and scalability, the relationship between the workload on different queue managers and the application servers to which they are connected is complex. You can contact your IBM representative to obtain expert advice.

The queue managers run on the same hosts as the application servers

In the subsequent figure:

- Application severs 1, 2 and 3 are part of the same WebSphere Application Server cluster.
- Application servers 1 and 3 are running on Host 1.
- Application server 2 is running on Host 2.
- Queue managers 1, 2 and 3 are part of the same WebSphere MQ cluster.
- Queue manager 1 is running on Host 1.
- Queue manager 2 is running on Host 2.
- Queue manager 3 is running on Host 3.
- Queue manager 3 is responsible for distributing messages between the cluster queues in a way that achieves workload balancing.
- The transport type for the connection is specified as “bindings”. A "bindings" connection is used when the application server and the queue manager are running on the same host. This is a cross-memory connection that is used to communicate with a queue manager. A bindings connection is also known as “call attach”.
  - Application servers 1 and 3 attach to queue manager 1 in bindings mode.
  - Application server 2 attaches to queue manager 2 in bindings mode.

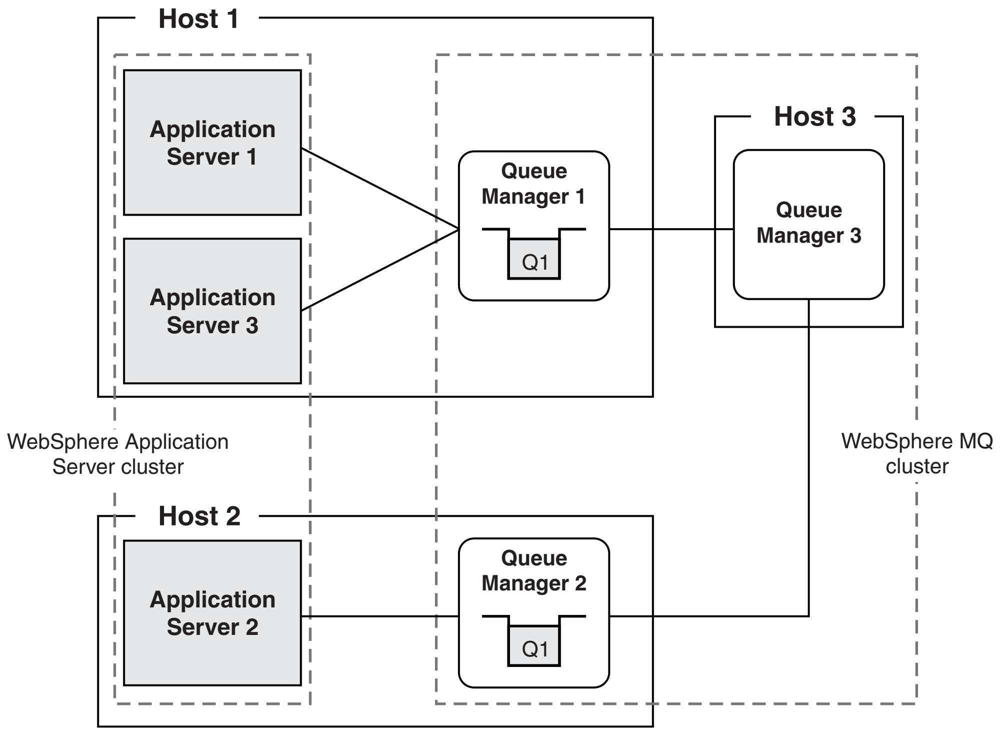


Figure 33. WebSphere Application Server clustering: bindings mode attachment to queue managers

If application server 1 fails:
Application server 3 can take over its workload because they are both attached to queue manager 1.

If application server 3 fails:
Application server 1 can take over its workload because they are both attached to queue manager 1.

If application server 2 fails:
You must restart it as soon as possible for the following reasons:

- Because no other application server is attached to queue manager 2 no other application server can take over its WebSphere MQ workload. The workload that was generated by application server 2 ceases. Other application servers in the cluster can, however, take over its external workload
- Queue manager 3 continues to distribute work between queue manager 1 and queue manager 2, even though the workload arriving at queue manager 2 cannot be taken on by application server 2.

  Note: If you choose not to restart, you can alleviate this situation by manually configuring Q1 on queue manager 2 so that the ability to put messages to it is inhibited. This results in all messages being sent to queue manager 1 where they are processed by the other application servers.

If queue manager 1 fails:
You must restart it as soon as possible for the following reasons:

- Messages that are on queue manager 1 when it fails are not processed until you restart queue manager 1.
- Because application servers 1 and 3 are not attached to queue manager 2, they cannot take on any of its workload.

- No new messages from WebSphere MQ applications are sent to queue manager 1, instead new messages are sent to queue manager 2 and consumed by application server 2.
- Because application servers 1, 2 and 3 are in the same WebSphere Application Server cluster, their non-WebSphere MQ workload continues to be distributed between them all, even though application servers 1 and 3 cannot use WebSphere MQ because queue manager 1 has failed.

Although this networking topology can provide availability and scalability, the relationship between the workload on different queue managers and the application servers with which they are connected is complex. You can contact your IBM representative to obtain expert advice.

Connecting WebSphere Application Server application servers to WebSphere MQ for z/OS with queue-sharing groups:

On z/OS systems, an application server can connect to a queue manager that is a member of a WebSphere MQ for z/OS queue-sharing group. You can configure the connection so that it selects a specific named queue manager, or you can configure it to accept any queue manager in the queue-sharing group.

Note: In this topic "application server" refers to an application server that is running on WebSphere Application Server and "queue manager" refers to a queue manager that is running on WebSphere MQ.

If you configure a connection to select a specific named queue manager, your options for providing high availability are like those for connecting to WebSphere MQ on other platforms. However, you can improve availability if you configure the connection to accept any queue manager in the queue-sharing group. In this situation, when the application server reconnects following a WebSphere MQ queue manager failure, the application server can accept connection to a different queue manager that has not failed.

A connection that you configure to accept any queue manager must only be used to access shared queues. A shared queue is a single queue that all queue managers in the queue-sharing group can access. It does not matter which queue manager an application uses to access a shared queue. Even if the same application instance uses different queue managers to access the same shared queue, this always produces consistent results.

These examples show two topology options for connecting to WebSphere MQ for z/OS to benefit from queue-sharing groups:
- The application servers and the queue managers run in the same logical partition (LPAR)
- The application servers and the queue managers run in different logical partitions (LPARs)

The application servers and the queue managers run in the same logical partition (LPAR)

The following figure shows a bindings mode connection from WebSphere Application Server to WebSphere MQ for z/OS. The figure shows the following configuration:
- Application servers 1 and 2 are part of a WebSphere Application Server cluster.
- Application server 1 is running in LPAR 1.
- Application sever 2 is running in LPAR 2.
- Queue managers 1 and 2 are members of a WebSphere MQ queue-sharing group that hosts a shared queue, Q1. The shared queue is located in a coupling facility.
- Queue manager 1 is running in LPAR 1.
- Queue manager 2 is running in LPAR 2.
- A "bindings" connection is used when the application server and the queue manager are running on the same host. This is a cross-memory connection is established to a queue manager running on the same host. A bindings connection is also known as "call attach".

- – Application server 1 and queue manager 1 are attached to each other in bindings mode.
- – Application server 2 and queue manager 2 are attached to each other in bindings mode.

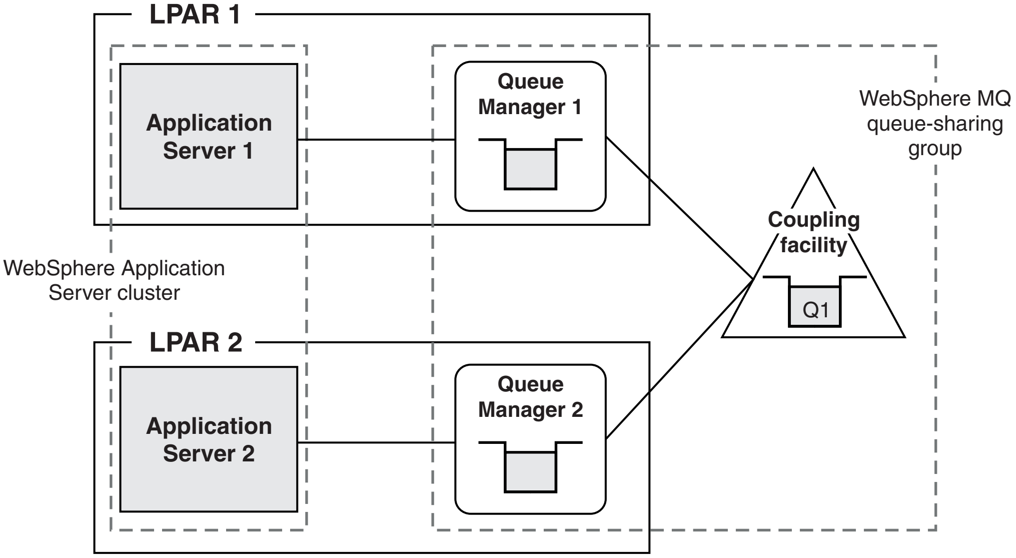


Figure 34. WebSphere Application Server with bindings mode connection to WebSphere MQ for z/OS

This networking topology can benefit from “pull” workload balancing if several application instances, including instances running in different LPARs, are processing messages from the same shared queue.

You can improve availability for this topology by using the z/OS Automatic Restart Manager (ARM) to restart failed application servers or queue managers. If a queue manager in an LPAR fails, ARM can restart an application server in a different LPAR, where the application server can connect to a running queue manager, instead of waiting for a restart of the queue manager that it was using previously. In the example used here, ARM can restart WebSphere Application Server application server 1 in LPAR 2, where it can connect to WebSphere MQ queue manager 2, instead of waiting for queue manager 1 to restart.

The application servers and the queue managers run in different logical partitions (LPARs)

The following figure shows a client mode connection from WebSphere Application Server to WebSphere MQ for z/OS. The figure shows the following configuration:

- Queue managers 1 and 2 are members of a WebSphere MQ queue-sharing group that hosts a shared queue, Q1. The shared queue is located in a coupling facility. The two queue managers run in different LPARs.
- A "client" connection is used when the application server and queue manager are running on different hosts. This is a TCP/IP network connection that is used to communicate with the queue manager. A client connection is also known as “socket attach”.
  - – Multiple application servers have a client mode (TCP/IP) connection to the queue managers. All the client mode connections are managed by the z/OS sysplex distributor, which selects either queue manager 1 or queue manager 2 for each connection request.

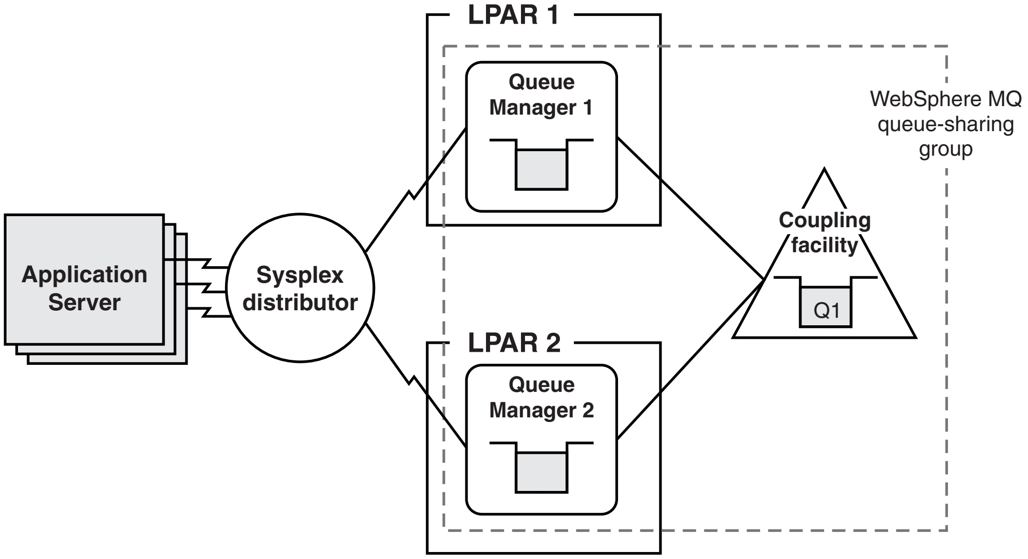


Figure 35. WebSphere Application Server with client mode connection to WebSphere MQ for z/OS

As with the bindings mode connection example, this networking topology can benefit from “pull” workload balancing if several application instances running in the same or different application servers are processing messages from the same shared queue.

The use of the z/OS sysplex distributor improves availability for this networking topology. If one of the queue managers fails, the z/OS sysplex distributor can connect applications running in the application servers to the other queue manager, without waiting for the failed queue manager to restart. In the example used here, if queue manager 1 fails, the z/OS sysplex distributor can select queue manager 2 for every connection request, until queue manager 1 restarts.

Note: In this networking topology, WebSphere MQ for z/OS GROUP units of recovery must be enabled on all the queue managers in the queue-sharing group. TCP/IP (client mode) connections that accept any queue manager use GROUP units of recovery. GROUP units of recovery are not supported by versions of WebSphere MQ for z/OS earlier than Version 7.0.1. Bindings mode connections do not require GROUP units of recovery.

WebSphere MQ messaging provider activation specifications

Activation specifications are used to configure inbound message delivery to message-driven beans (MDBs) running inside WebSphere Application Server. They supersede message listener ports, which are now a stabilized function.

Activation specifications and message-driven beans

Activation specifications are the standardized way to manage and configure the relationship between an MDB running in WebSphere Application Server and a destination withinWebSphere MQ. They combine the configuration of connectivity, the Java Message Service (JMS) destination and the runtime characteristics of the MDB, within a single object.

Message-driven beans are a special class of Enterprise Java Bean (EJB). They enable Java Platform, Enterprise Edition (JEE) applications to process messages asynchronously, with WebSphere Application Server managing the transactionality and concurrency of the application.

The following figure shows how an activation specification can be used to link a WebSphere MQ queue manager destination to an MDB running within WebSphere Application Server. The process of delivering a message from a client to an MDB via an WebSphere MQ messaging provider activation specification occurs in this way:

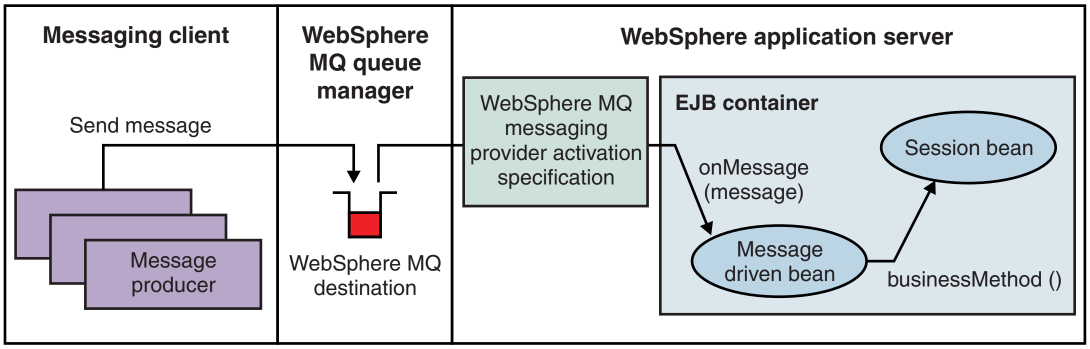


Figure 36. WebSphere MQ messaging provider activation specification in action

- A messaging client, either running in a stand-alone process or within an application server environment, sends a message using JMS (or any other messaging API, such as MQI) to a WebSphere MQ queue or topic defined in a WebSphere MQ queue manager.
- A WebSphere MQ activation specification is configured to listen on that destination for messages. When the new message is detected, it is removed from the destination (potentially under an XA transaction).
- The message is then passed to an MDB that has been configured to use the activation specification through its onMessage method.
- The MDB uses the information in the message to perform the relevant business logic.

Activation specifications compared with listener ports

Versions of WebSphere Application Server earlier than Version 7.0, use listener ports to define the association between a connection factory, a destination, and an MDB.

Activation specifications supersede the use of listener ports, which became a stabilized feature in WebSphere Application Server Version 7.0 (for more information, see “Stabilized features” on page 1137). There are several advantages to using activation specifications over listener ports:

- Activation specifications are simple to configure, because they only require two objects: the activation specification and a message destination. Listener ports require three objects: a connection factory, a message destination, and the message listener port itself.
- Activation specifications are not limited to the server scope. They can be defined at any administrative scope in WebSphere Application Server. Message listener ports must be configured at the server scope. This means that each server in a node requires its own listener port. For example, if a node is made up of three servers, three separate listener ports must be configured. Activation specifications can be configured at the node scope, so in the example only one activation specification would be needed.
- Activation specifications are part of the Java Platform, Enterprise Edition Connector Architecture 1.5 and 1.6 standards specification (JCA 1.5 and 1.6). Listener port support in WebSphere Application Server makes use of the application server facilities interfaces defined in the JMS specification, but is not part of any specification itself.

It is still possible to use message listener ports to deliver messages to an MDB using the WebSphere MQ messaging provider. There are certain scenarios in which the use of listener ports is still preferable to using activation specifications. This usually is the case with configurations in which some of the servers are running on versions of WebSphere Application Server earlier than WebSphere Application Server Version 7.0. It is possible to configure both message listener ports (which make use of WebSphere MQ messaging provider resources) and WebSphere MQ messaging provider activation specifications at the same time. For more information, see “Message-driven beans, activation specifications, and listener ports” on page 143.

To assist in migrating listener ports to activation specifications, the WebSphere Application Server administrative console provides a **Convert listener port to activation specification** wizard on the Message listener port collection panel. This allows you to convert existing listener ports into activation specifications. However, this function only creates a new activation specification with the same configuration used by the listener port. It does not modify application deployments to use the newly created activation specification.

Enhanced features of the WebSphere MQ messaging provider

The WebSphere MQ messaging provider enables WebSphere Application Server applications and clients to connect to and use WebSphere MQ resources in a JMS-compliant manner. This provider includes the enhanced features described in this topic.

Overview

The WebSphere MQ messaging provider has enhanced administrative options supporting the following functions:

- “WebSphere MQ channel compression”
- “WebSphere MQ client channel definition table”
- “Client channel exits” on page 397
- “Transport-level encryption by using SSL” on page 397

WebSphere MQ channel compression

Data sent over the network between WebSphere Application Server and WebSphere MQ can be compressed, reducing the amount of data that is transferred. Channel compression can be beneficial in the following situations:

- If a cost is incurred that is proportional to the amount of data transferred over a network. For example, nodes in a network might span a leased line for which a utilization charge is applied.
- If the rate at which messaging data can be transferred across a network is the limiting factor in the performance of an application.
- If compressing the data might reduce the cost of its encryption and decryption.

To use WebSphere MQ channel compression, configure the message compression properties of an existing connection factory or activation specification. For more information, see the appropriate step within Configuring JMS resources for the WebSphere MQ messaging provider.

For more information, see the WebSphere MQ topic *Channel compression* in the WebSphere MQ information center that is part of the WebSphere MQ library.

WebSphere MQ client channel definition table

The client channel definition table reduces the effort required to configure a connection to a queue manager. Your WebSphere MQ administrator can create a single table of all the WebSphere MQ channels supported by queue managers in the enterprise, then in WebSphere Application Server you configure a connection to a queue manager by identifying the client channel definition table and providing any additional information not already contained within the table.

You can also use the client channel definition table to provide a basic failover capability, by specifying that a connection is attempted against several queue managers listed in the table. Each suitable channel definition is tried in turn until a queue manager connection is successfully established.

You can use the client channel definition table, with WebSphere MQ messaging provider activation specifications and connection factories, to select the client channel definition to use when establishing a connection to WebSphere MQ. The table can be configured to select from a number of queue managers, depending on their availability.

When you use a client channel definition table, note the following restrictions:

- If your client channel definition table can select from more than one queue manager, you might not be able to recover global transactions. Activation specifications and connection factories that specify a client channel definition table must either do so without ambiguity as to the target queue manager, or must avoid using the resources with applications that enlist in global transactions.
- If your client channel definition table contains channel definitions that reference native WebSphere MQ channel exits, the use of these channel definitions is not supported in the WebSphere Application Server environment.

For more information about client channel definition tables, see the developerWorks article WebSphere MQ V6 Java and JMS clients and the client channel definition table, and the WebSphere MQ topic Client channel definition table.

To use a client channel definition table, specify it when you create a new activation specification or connection factory.

Client channel exits

Client channel exits are pieces of Java code that you develop, and that run in the application server at key points during the life cycle of a WebSphere MQ channel. Your code can change the runtime characteristics of the communications link between the WebSphere MQ messaging provider and the WebSphere MQ queue manager.

Note: Only client channel exits written in Java are supported for use within the WebSphere Application Server environment.

For more information about client channel exits, see the WebSphere MQ topic Channel exit programs. For a list of the channel exits that work with the WebSphere MQ messaging provider, see the client connection channel row of the table in the WebSphere MQ topic What are channel exit programs?.

To use client channel exits, configure the client transport properties of an existing connection factory or activation specification.

Transport-level encryption by using SSL

Transport-level encryption by using SSL is the supported way to configure SSL for JMS resources associated with the WebSphere MQ messaging provider. The SSL configuration is associated with the communication link for the connection factory or activation specification. You either define the SSL information in the connection factory, or your WebSphere MQ administrator defines the SSL information in an associated client channel definition table.

Strict message ordering with the WebSphere MQ messaging provider and message-driven bean (MDB) applications

Message ordering is important to some asynchronous messaging applications; that is, it is important to process messages in the same order that the producer sends them. If this type of message ordering is important to your application, your design must take it into account.

For example, a messaging application that processes seat reservations might have producer components and a consumer component. A producer component sends a message to the consumer component when a customer reserves a seat. If the customer cancels the reservation then the producer (or possibly a different producer) sends a second message. Typically, the consumer component must process the first message (which reserves the seat) before it processes the second message (which cancels the reservation).

Some applications use a synchronous (request-response) pattern where the producer waits for a response to each message before it sends the next message. In this type of application, the consumer controls the order in which it receives the messages and can ensure that this is the same order as the producer or producers send them. Other applications use an asynchronous (fire and forget) pattern where the producer sends messages without waiting for responses. Even for this type of application, order is usually preserved; that is, a consumer can expect to receive messages in the same order as the producer or producers send them, especially when there is a significant time between sending consecutive messages. However your design must consider factors that can disrupt this order.

The order of messages is disrupted if your application sends messages with different priorities (higher priority messages can overtake lower priority messages) or if your application explicitly receives a message other than the first by specifying message selectors. Parallel processing and error or exception processing can also affect message ordering.

Strict message ordering using non-ASF listener ports:

Strict message ordering can be achieved when deploying message driven bean applications to the WebSphere MQ messaging provider when no special facilities have been coded into the application to handle messages arriving out of order.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

The following assumptions have been made in this scenario:

- The message-driven bean (MDB) application is transactional.
- The back-out threshold (BOTHRESH) on the WebSphere MQ queue has been set to `0`.

WebSphere Application Server configuration for ordered delivery

- Non-ASF mode must be activated by specifying a non-zero timeout value for the **`NON.ASF.RECEIVE.TIMEOUT`** WebSphere MQ message listener service custom property.
- The **Maximum sessions** setting for the listener port must be set to `1`.
- A non-ASF listener port with **Maximum sessions** set to `1` has a single thread running inside the application server retrieving messages. When a message arrives the thread immediately delivers it to the MDB.
- The queue manager regards this thread as a single application retrieving messages, therefore messages are processed in sequence.

Messages can be delivered out of order with this deployment during a transaction recovery

A specific set of events must occur in a specific order for this scenario to be encountered, and as such it is uncommon. However, if ordered message delivery is critical to the operation of your application, the you must consider it.

- Out of order message delivery can occur with this deployment option during recovery from a failure of one of the following components:

  - The application server hosting the MDB
  - The WebSphere MQ queue manager
  - A network connecting the application server and queue manager
- If one of these components fails in the middle of a two-phase commit of an MDB transaction, the application server transaction manager reestablishes its connection to the queue manager to resolve the transaction when the component is available again.
- This recovery process is asynchronous, and it is possible for delivery of new messages to the MDB to begin before the transaction recovery process is complete. If the outcome of the transaction recovery is to roll back the transaction, then the message will be returned to the WebSphere MQ queue and re-delivered to the application, possibly after new messages have already been delivered.

Considerations for clustered deployment

When you are using non-ASF listener ports you can set the default share **(DEFSOPT)** option on the queue to `exclusive`. If you choose this option when you are performing a clustered deployment of an application, all but one of the cluster members fail to start their listener ports. The cluster members generate a `2042MORC_OBJECT_IN_USE` exception, in a `WMSG0057E` message.

When this exception occurs you can then establish automatic failover for the application by configuring the following message listener service custom property in WebSphere Application Server:

MAX.RECOVERY.RETRIES

Configure a high value **MAX.RECOVERY.RETRIES** on the message listener services of all the servers in the cluster. The maximum value for **MAX.RECOVERY.RETRIES** is `2147483647`.

The **MAX.RECOVERY.RETRIES** message listener service custom property must be accompanied by a suitable **MAX.RECOVERY.INTERVAL** message listener service custom property. The maximum amount of time a listener port can retry without being manually stopped and restarted is 2147483647 times the value specified for **MAX.RECOVERY.INTERVAL**. In this configuration each cluster member continuously attempts to start its listener port, until the active cluster member stops and the queue manager allows it to connect as a single exclusive consumer.

Strict message ordering using activation specifications or ASF listener ports connected to WebSphere MQ Version 7.x:

Strict message ordering can be achieved when deploying message driven bean applications to the WebSphere MQ messaging provider when no special facilities have been coded into the application to handle messages arriving out of order.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

The following assumptions have been made in this scenario:

- The message-driven bean (MDB) application is transactional.
- The back-out threshold (BOTHRESH) on the WebSphere MQ queue has been set to `0`.
- You are using WebSphere MQ Version 7.0 or later.

WebSphere Application Server configuration for ordered delivery

- The WebSphere MQ queue manager must be running on WebSphere MQ Version 7.0 or later.

- The connection to the queue manager must use the WebSphere MQ messaging provider normal mode. See the *Rules for selecting the WebSphere MQ messaging provider mode* topic of the WebSphere MQ information center.
- If you are using listener ports **Maximum sessions** on the listener ports in WebSphere Application Server must be set to `1`.
- If you are using activation specifications **Maximum server sessions** on the activation specifications in WebSphere Application Server must be set to `1`.

Important information about this configuration

- ASF listener ports and WebSphere MQ activation specifications contain two separate parts, which together perform message delivery. These two parts are seen as separate applications by the queue manager:
  - Part one detects messages as they arrive, but does not consume them. Instead it dispatches them to the second part.
  - Part two is a server session pool which allocates a thread to process the message within the application's transaction, and deliver it to the onMessage() method of the MDB.
- From Version 7.0, WebSphere MQ provides a push model for detection of the messages, which is more efficient than the polling model used in previous versions of WebSphere MQ, and provides better ordering of message under normal operation.

Circumstances in which messages can be delivered out of order

Messages can be delivered out of order with this configuration in the following circumstances:

- After a transaction rollback, the next message available on the queue might be delivered before the rolled back message is re-delivered:
  - For an ASF listener port, setting **Maximum retries** to `zero` prevents out of order delivery after a rollback by stopping the listener port when a rollback occurs. However, the listener port must then be restarted manually.
  - For an activation specification, selecting **Stop endpoint if message delivery fails** and setting **Number of sequential delivery failures before suspending endpoint** to `0` prevents out of order delivery after a rollback by pausing the message endpoint when a rollback occurs. However, the message endpoint for the MDB must then be resumed manually. For more information, see the WebSphere MQ information center.
- Messages can be delivered out of order during a transaction recovery:

  Note: A specific set of events must occur in a specific order for this scenario to be encountered, and as such it is uncommon. However, if ordered message delivery is critical to the operation of your application, the you must consider it.
  - Out of order message delivery can occur with this deployment option during recovery from a failure of one of the following components:
    - The application server hosting the MDB
    - The WebSphere MQ queue manager
    - A network connecting the application server and queue manager
  - If one of these components fails in the middle of a two-phase commit of an MDB transaction, the application server transaction manager reestablishes its connection to the queue manager to resolve the transaction when the component is available again.
  - This recovery process is asynchronous, and it is possible for delivery of new messages to the MDB to begin before the transaction recovery process is complete. If the outcome of the transaction recovery is to roll back the transaction, then the message will be returned to the WebSphere MQ queue and re-delivered to the application, possibly after new messages have already been delivered.

Considerations for a clustered deployment

- You must activate the MDB on one cluster member only, as the application server does not have a facility which can manage this activation automatically.
- You can set the startup state of listener ports to stopped, separately to setting the startup state of an application.
- You can manually start and stop applications, ASF listener ports, and message endpoints with MBean interfaces by using wsadmin scripting, or by using the com.ibm.websphere.management.AdminClient interfaces from Java code.

Strict message ordering using activation specifications or ASF listener ports connected to WebSphere MQ Version 6.0:

Strict message ordering can be achieved when deploying message driven bean applications to the WebSphere MQ messaging provider for when no special facilities have been coded into the application to handle messages arriving out of order.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

The following assumptions have been made in this scenario:

- The message-driven bean (MDB) application is transactional.
- The back-out threshold (BOTHRESH) on theWebSphere MQ queue has been set to `0`.
- You are using WebSphere MQ Version 6.0.

WebSphere Application Server configuration for ordered delivery

- If you are using listener ports **Maximum sessions** on the listener ports in WebSphere Application Server must be set to `1`.
- If you are using activation specifications **Maximum server sessions** on the activation specifications in WebSphere Application Server must be set to `1`.

Important information about this configuration

- ASF listener ports and WebSphere MQ activation specifications contain two separate parts, which together perform message delivery. These two parts are seen as separate applications by the queue manager:
  - One part detects messages as they arrive, but does not consume them. Instead it dispatches them to the second part.
  - Part two is a server session pool which allocates a thread to process the message within the application's transaction, and deliver it to the onMessage() method of the MDB.

  Note: When an ASF listener port or activation specification is connected to a WebSphere MQ Version 6.0 or earlier queue manager, a less efficient polling mechanism (based on a browse cursor) is used to detect messages on the queue.

Circumstances in which messages can arrive out of order

Messages can arrive out of order with this deployment in the following circumstances:

- Messages can be delivered out of order during a transaction recovery:

Note: A specific set of events must occur in a specific order for this scenario to be encountered, and as such it is uncommon. However, if ordered message delivery is critical to the operation of your application, the you must consider it.

  - Out of order message delivery can occur with this deployment option during recovery from a failure of one of the following components:
    - The application server hosting the MDB
    - The WebSphere MQ queue manager
    - A network connecting the application server and queue manager
  - If one of these components fails in the middle of a two-phase commit of an MDB transaction, the application server transaction manager reestablishes its connection to the queue manager to resolve the transaction when the component is available again.
  - This recovery process is asynchronous, and it is possible for delivery of new messages to the MDB to begin before the transaction recovery process is complete. If the outcome of the transaction recovery is to roll back the transaction, then the message will be returned to the WebSphere MQ queue and re-delivered to the application, possibly after new messages have already been delivered.
- After a transaction rollback, the next message available on the queue might be delivered before the rolled back message is re-delivered:
  - For an ASF listener port, setting **Maximum retries** to `zero` prevents out of order delivery after a rollback by stopping the listener port when a rollback occurs. However, the listener port must then be restarted manually.
  - For an activation specification, selecting **Stop endpoint if message delivery fails** and setting **Number of sequential delivery failures before suspending endpoint** to `0` prevents out of order delivery after a rollback by pausing the message endpoint when a rollback occurs. However, the message endpoint for the MDB must then be resumed manually. For more information, see the WebSphere MQ information center.
- During normal operation, when multiple threads are sending messages to the destination (for different sequences) using transactions:
  - This behavior is due to the operation of the WebSphere MQ browse cursor.
  - When a message is committed to a WebSphere MQ queue, while another message sent to the destination is uncommitted (within a transaction that has not yet completed), the browse cursor moves onto the newer message on the queue and does not automatically return to the earlier message when it is eventually committed. This can cause messages to appear in the queue following the browse cursor.
  - If this scenario occurs, newer messages within a sequence might be delivered to the MDB before the WebSphere MQ messaging provider re-scans the queue and detects the message that has appeared behind the browse cursor.
- If the WebSphere MQ queue being monitored by an activation specification or ASF listener port has the Message delivery sequence attribute (MSGDLYSEQ) set to priority, message ordering can fail due to the following reasons:
  - Messages of a lower priority might be delivered ahead of messages of a higher priority, when messages of multiple priorities are sent to a queue, this behavior is due to the operation of the WebSphere MQ browse cursor. The browse cursor moves through all available messages at the highest priority, and then moves to lower priority messages. If higher priority messages arrive when the browse cursor is currently browsing lower priority messages, those higher priority messages might not be delivered until after all lower priority messages on the queue have been delivered.
  - ASF listener ports or activation specifications that browse queues that have **Message delivery sequence** set to `FIFO` do not see this issue, as WebSphere MQ orders the messages on the queue in the order in which they arrive, rather than ordering them by priority.

Considerations for a clustered deployment

- You can activate the MDB on one cluster member only, as the application server does not have a facility which can manage this activation automatically.

- You can set the startup state of listener ports to stopped, separately to setting the startup state of an application.
- You can manually start and stop applications, ASF listener ports, and message endpoints with MBean interfaces by using wsadmin scripting, or by using the com.ibm.websphere.management.AdminClient interfaces from Java code.

WebSphere MQ custom properties

WebSphere Application Server supports the use of custom properties to define WebSphere MQ properties. This is useful because it enables WebSphere Application Server to work with later versions of WebSphere MQ that might have properties that are not available in the WebSphere Application Server administrative console.

For WebSphere Application Server Version 7.0 or later, the custom properties that you define are validated by the WebSphere MQ resource adapter contained in WebSphere Application Server. In earlier releases, this was done within WebSphere Application Server itself, and then by the WebSphere MQ client jar files. If you have defined a property that is not valid for WebSphere MQ, the WebSphere MQ resource adapter creates an exception, which is caught by WebSphere Application Server, and logged in the `Systemout.log` and `SystemErr.log` files. Examples of error messages are given at the end of this topic.

When a later version of WebSphere MQ is available that is supported by the WebSphere Application Server installation, new WebSphere MQ properties might be created that are not known to WebSphere Application Server. You can configure these as custom properties through WebSphere Application Server so that they are recognized by the WebSphere MQ resource adapter. You can also configure WebSphere Application Server to point to the WebSphere MQ resource adapter in the external JMS provider, as described in Configuring the WebSphere MQ messaging provider with native libraries information.

For information on valid values for WebSphere MQ properties, refer to the *Using Java* and *System Administration* sections of the WebSphere MQ information center.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Error message example

The exception created by the resource adapter contains error messages similar to the following example:

```
[09/02/06 15:40:06:377 GMT] 0000000a ContainerImpl E   WSVR0501E: Error creating
component null [class com.ibm.ws.runtime.component.ApplicationServerImpl]
com.ibm.ws.exception.RuntimeWarning: com.ibm.ws.runtime.component.binder.
ResourceBindingException: invalid configuration passed to resource binding logic.
REASON: Failed to create connection factory: Error raised constructing AdminObject,
error code: XAQCF PropertyName : XAQCF PropertyName
```

where `PropertyName` is the name of the invalid property.

WebSphere MQ messages

A WebSphere MQ message usually contains a message descriptor, one or more message headers, and a message payload. WebSphere MQ provides programming interfaces that can help your applications to process WebSphere MQ messages.

Components of a WebSphere MQ message

A WebSphere MQ message contains one or more of the following components:

Message descriptor

The message descriptor contains standard message properties, applicable to all WebSphere MQ messages. For example, the message descriptor contains the message identifier and the correlation identifier (similar to the JMSMessageID and the JMSCorrelationID). Every WebSphere MQ message contains a message descriptor (MQMD).

Message headers

A message header usually contains additional message properties applicable to particular types of message. For example, WebSphere MQ messages sent to the CICS bridge include a CICS bridge header (MQCIH). One exception is the "WebSphere MQ rules and formatting header 2" (MQRFH2), which can be used to contain message properties for various message types, including JMS message properties. Which headers (if any) a WebSphere MQ message contains depends on the intended recipient.

Message Properties

WebSphere MQ message properties were introduced with the WebSphere MQ Version 7.0 JMS client. A WebSphere MQ message property is data associated with a message, consisting of a textual name and a value of a particular type. WebSphere MQ message properties are used by message selectors to filter publications to topics or to selectively get messages from queues. WebSphere MQ message properties can be used to include business data or state information without having to store it in the application data. Applications do not have to access data in the MQMD or MQRFH2 headers because the fields in these data structures can be accessed as WebSphere MQ message properties.

Message payload

The message payload is the data (if any) that follows the last message header, or that follows the MQMD if there are no message headers.

For more information about WebSphere MQ messages, see the *Application Programming Guide* and *Application Programming Reference* sections of the WebSphere MQ information center.

WebSphere MQ implementation of a JMS message

WebSphere MQ provides a programming interface called the Message Queue Interface (MQI). This interface allows applications to process the components of a WebSphere MQ message using a variety of programming languages. WebSphere MQ also provides a JMS programming interface which allows applications to process a WebSphere MQ message as a JMS message. In this case the JMS message body is usually the WebSphere MQ message payload. However the JMS message body can include WebSphere MQ message headers or, if you are using WebSphere MQ Version 7.x, WebSphere MQ message properties, or both. For details of WebSphere MQ support for JMS, see the *Using Java* section of the WebSphere MQ information center.

How messages are passed between service integration and a WebSphere MQ network

When you program a WebSphere Application Server application that interoperates with WebSphere MQ through the default messaging provider and a service integration bus, service integration automatically handles most aspects of the conversion and mapping for messages. Understanding how this process works and the differences between the two environments helps you to program and troubleshoot your applications more effectively.

The basic message conversion process is the same whether the service integration bus is interoperating with WebSphere MQ through a WebSphere MQ link or with a WebSphere MQ server. For further information about different types of messaging with WebSphere MQ links and WebSphere MQ servers, read whichever of the following sections applies to your system's architecture:

- Interoperation using a WebSphere MQ link
- Interoperation using a WebSphere MQ server

Differences between service integration and a WebSphere MQ network

Applications can use both service integration and WebSphere MQ to convey messages. Service integration messaging uses messaging engines, whereas WebSphere MQ uses queue managers.

WebSphere MQ is a stand-alone messaging and queuing system, and is not a part of an application server. In WebSphere MQ, messaging and queuing services are provided by queue managers. An application attaches to a queue manager and uses an application programming interface to get messages on and from queues. One of these application interfaces is the Java Messaging Service (JMS) API. Applications can attach directly to a queue manager using a call interface or indirectly using a TCP/IP socket connection. The TCP/IP socket connection which an application uses to attach to a queue manager is called an MQI channel. The application uses the same programming interface for both direct (bindings mode) and indirect (client mode) attachments.

Service integration is part of WebSphere Application Server. In service integration, messaging and queuing services are provided by messaging engines (ME). A service integration messaging engine runs in a WebSphere Application Server server. A service integration messaging engine is similar to a WebSphere MQ queue manager plus its associated message channel agent (MCA), which is used to move messages from one queue manager to another. However, unlike a queue manager, a messaging engine also includes transformation and routing capabilities.

A WebSphere Application Server application connects to a messaging engine using JMS services, and uses the JMS application programming interface to send and receive messages from destinations. A JMS destination is similar to a WebSphere MQ queue or topic. A service integration messaging engine uses WebSphere Application Server communication capabilities for connecting clients outside the WAS server where it runs, and for communicating with other messaging engines. Service integration messaging engines provide services for transformation and routing, and support publish/subscribe messaging. A separate message broker is not required.

WebSphere MQ applications consume messages from queues that are locally defined on a queue manager or (for WebSphere MQ for z/OS) queue-sharing group. In service integration, the equivalent component to a WebSphere MQ locally-defined queue is a queue point on the local messaging engine. In service integration there is no similar restriction imposed on the queue point and the location in the bus where the consuming application is connected.

The JMS API is available to messaging applications in both WebSphere Application Server and WebSphere MQ. WebSphere MQ also has a native API called the Message Queue Interface (MQI). The JMS send and receive interfaces are similar to the MQI put and get interfaces.

Each WebSphere MQ queue manager usually has a dead-letter queue (also known as the undelivered message queue) defined for it. Messages are put on this queue if they cannot be delivered to their intended destination. In WebSphere Application Server service integration, the equivalents to dead-letter queues are exception destinations. A default exception destination is automatically created for each messaging engine. If messages cannot be delivered, messages are put on the specific exception destination for the queue, if it exists, or on the default exception destination.

How service integration converts messages to and from WebSphere MQ format

Messages are converted between WebSphere MQ format and service integration format as they flow between the two systems.

Exchanging messages between JMS programs through service integration and WebSphere MQ

Usually, you do not have to be aware of conversion between message formats to exchange JMS messages between service integration and WebSphere MQ, because service integration performs the appropriate conversion automatically, including character and numeric encoding. However, you might have to learn about message conversion if your JMS applications do not behave as expected, or if your service integration configuration includes JMS programs or mediations that process messages to or from non-JMS WebSphere MQ programs.

If your service integration applications exchange MapMessage objects with WebSphere MQ applications, you might have to specify a non-default map message encoding format.

When service integration converts messages to and from WebSphere MQ format

Service integration converts a service integration message into a WebSphere MQ message in the following circumstances:

- When service integration sends a message to WebSphere MQ by using a WebSphere MQ link.
- When a service integration mediation places the message on a queue point that is a WebSphere MQ queue.
- When a service integration application sends the message to a destination where the mediation point (if any) or the queue point (if there is no mediation) is a WebSphere MQ queue.

Service integration converts a WebSphere MQ message into a service integration message in the following circumstances:

- When WebSphere MQ sends the message to a service integration bus by using a WebSphere MQ link.
- When a service integration mediation receives the message from a mediation point that is a WebSphere MQ queue.
- When a service integration application receives the message from a destination where the queue point is a WebSphere MQ queue.

When you use the WebSphere MQ messaging provider, there is no conversion between WebSphere MQ format and service integration format.

Overview of message conversion

When service integration converts a message to WebSphere MQ format, it usually constructs a WebSphere MQ message descriptor (MQMD), a rules and formatting header 2 (MQRFH2), and a message payload:

- **Message descriptor (MQMD)**: Service integration sets fields in the MQMD based on the service integration message header fields and properties; these include JMS message header fields and properties applicable to the message. Service integration always constructs an MQMD.
- **Rules and formatting header 2 (MQRFH2)**: Service integration sets fields in the MQRFH2 based on the service integration message header fields and properties. Some WebSphere MQ applications cannot process messages that contain an MQRFH2. To simplify interoperation, you can configure service integration to omit the MQRFH2 from messages for applications that cannot process the MQRFH2. However, be aware that when service integration omits the MQRFH2, it discards the corresponding service integration header fields and properties.
- **Message payload**: Service integration uses the body of the service integration message (if any) as the payload of the WebSphere MQ message.

When service integration converts a message from WebSphere MQ format:

- It sets the service integration message header fields and properties from the MQMD and (if present) the MQRFH2 in the WebSphere MQ message.
- It sets the service integration message body to the contents (if any) of the WebSphere MQ message that immediately follow the MQRFH2.
- If the message contains other headers, instead of, or as well as, the MQRFH2 header, those headers are treated as part of the JMS message body and the JMS message becomes a bytes message.

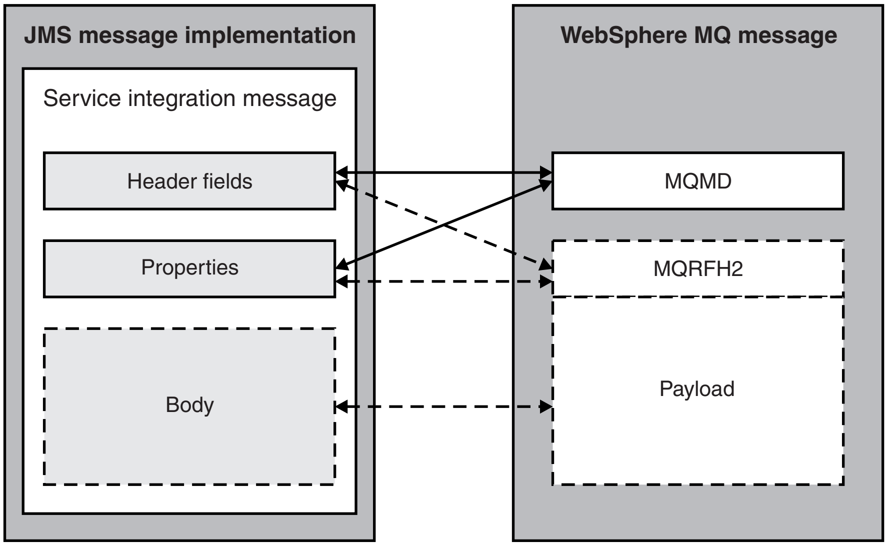


Figure 37. Message conversion to and from WebSphere MQ format

For reference information about the mappings for specific message header fields and properties between a service integration bus and WebSphere MQ, see the topics listed in the related reference. To help you program applications that interoperate with WebSphere MQ, these topics describe how the message formats are mapped between service integration messages and WebSphere MQ messages.

How to address bus destinations and WebSphere MQ queues

To understand how to access a service integration bus destination from WebSphere MQ, and a WebSphere MQ queue from a service integration bus, it is important to understand the different conventions that govern how these two resources are addressed.

For queue-type destinations, WebSphere MQ has a two-level addressing structure:

- queue manager name
- queue name

The equivalents for the service integration bus are:

- bus name
- destination name (identifier)

In WebSphere MQ, the queue manager name and queue name are each limited to 48 characters, and use of certain characters is restricted. For more information, see WebSphere MQ naming restrictions. The service integration bus equivalents do not have these restrictions, so (for example) messages from a WebSphere MQ application sent to a bus destination with a name longer than 48 characters must have some means of using the shorter name (used in WebSphere MQ) to address the longer name (used in the service integration bus). The service integration bus uses an alias destination to map between the shorter name and the long name. Similarly, an alias can also be used to send a message from a WebSphere Application Server application by using a long name (greater than 48 characters) and route it to a WebSphere MQ queue. For more information about alias destinations, see “Foreign destinations and alias destinations” on page 555.

Service integration `queue@queueManager` notation for WebSphere MQ queues

When service integration sends a message across a WebSphere MQ link, it must know the foreign bus that corresponds to the gateway queue manager or queue-sharing group; and when the send-to queue is defined in a different queue manager or queue-sharing group (not the gateway), service integration must know the location of the send-to queue so that it can save the correct name in the MQXQH **RemoteQMgrName** field. One way to achieve this is to define two foreign buses, an indirectly-connected bus (where the queue is defined) and a directly-connected bus (the gateway).

The following figure shows an example of this. In the figure, the target queue for a message is Q2 in queue manager QM2. The service integration configuration in the local bus defines QM2 as an indirectly-connected foreign bus and QM1 as the directly-connected intermediate bus. It defines Q2 as a foreign destination with bus name QM2 and destination name (identifier) Q2. The service integration configuration for the local bus does not include any information about the connection between QM1 and QM2.

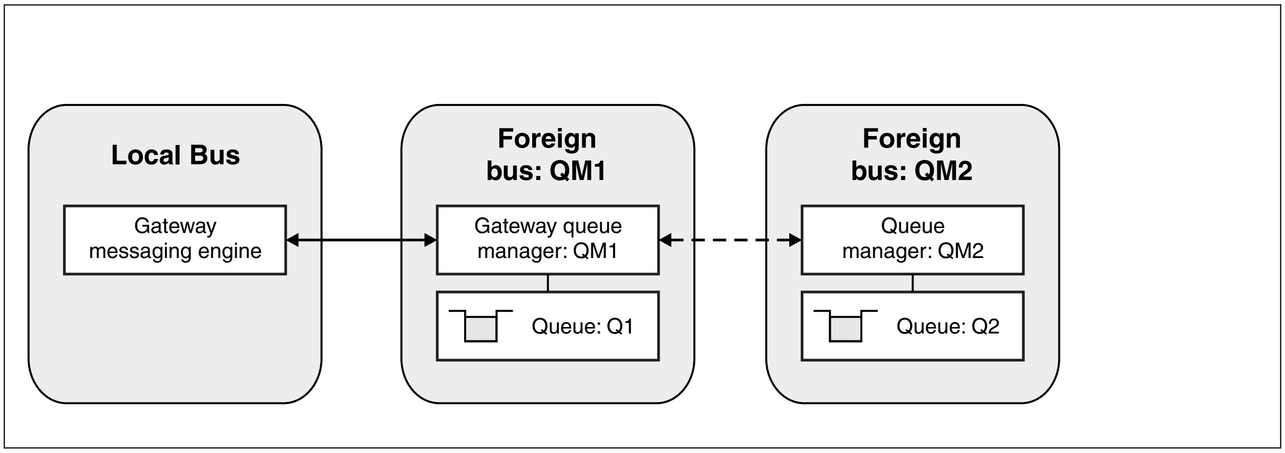


Figure 38. Addressing a WebSphere MQ queue in an indirectly connected foreign bus

Accessing a foreign WebSphere MQ queue in this way works perfectly well. However, when there are a large number of queue managers or queue-sharing groups that connect to the service integration bus through one gateway, you might find it inconvenient to define every one of them as an indirectly-connected foreign bus. Therefore service integration supports the following special destination name format for WebSphere MQ queues that includes both the queue name and the queue manager name joined by the at-sign (@): *queue@queueManager*. Using this special format, you do not have to define a separate indirectly-connected foreign bus for service integration because the name is part of the service integration destination name.

The following figure shows an example of this. In the figure, the target queue for a message is Q2 in queue manager QM2. The service integration configuration in the local bus does not define QM2 as a foreign bus. It defines Q2 as a foreign destination with bus name QM1 and destination name (identifier) Q2@QM2. The service integration configuration for the local bus does not include any information about the connection between QM1 and QM2.

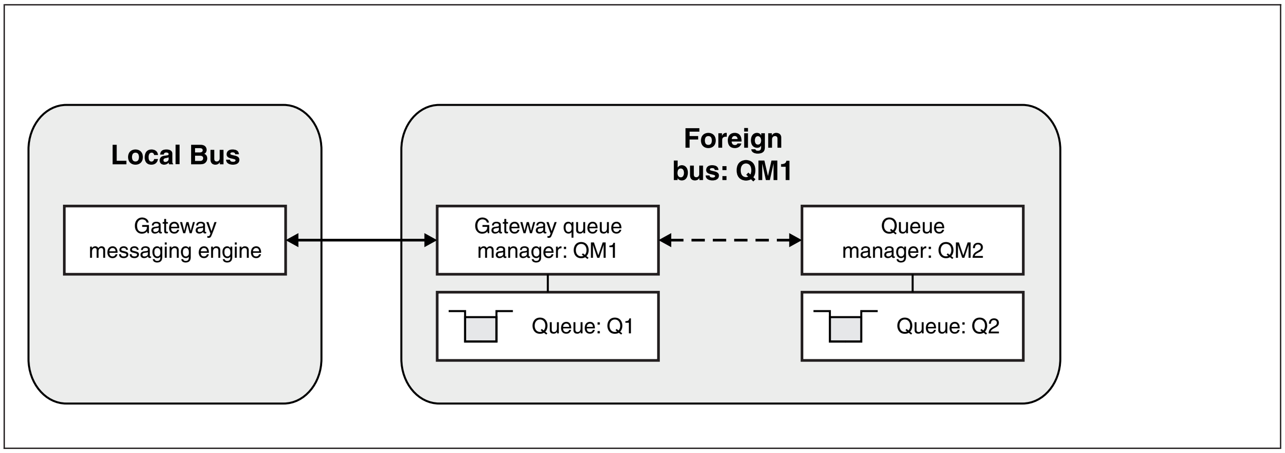


Figure 39. Addressing a WebSphere MQ queue by using `queue@queueManager` *notation*

Automatic mapping of the JMSReplyTo field of a JMS message

There are two fields in the JMS API that are used for sharing information about the destination to which a message is sent (JMSDestination) and the destination to which replies should be sent (JMSReplyTo). The JMSReplyTo field of a JMS message passing from a service integration bus to WebSphere MQ (or from WebSphere MQ to a service integration bus) is automatically mapped so that a consuming application in WebSphere MQ can reply to the original WebSphere Application Server application.

JNDI namespaces and connecting to different JMS provider environments

Interoperation with other JMS systems and clients is more straightforward if your messaging application connections are built using a connection factory and stored in a JNDI namespace. The JNDI namespace insulates your application from provider-specific information, and there are no differences that are significant for programming messaging applications.

The Java Naming and Directory Interface (JNDI) API enables JMS clients to look up configured JMS objects. By delegating all the provider-specific work to administrative tasks for creating and configuring these objects, the clients can be completely portable between environments. In addition, the applications are easier to administer because they have no specific administrative values embedded in their code.

There are two types of JMS administered objects:

- ConnectionFactory - the object a client uses to create a connection with a provider.
- Destination - the object a client uses to specify the destination for messages it is sending, and the source of messages it receives

The messaging environment to which the application connects depend on the implementation type of the ConnectionFactory object that is obtained from JNDI. For example, if the object is a WebSphere Application Server default messaging ConnectionFactory object, then a connection is made to the same service integration bus.

Interoperation using a WebSphere MQ link

A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network.

A WebSphere MQ link enables WebSphere Application Server applications to send point-to-point messages to WebSphere MQ queues, which are defined as destinations in the service integration bus. The link also enables WebSphere MQ applications to send point-to-point messages to destinations in the service integration bus, which are defined as remote queues in WebSphere MQ. With a publish/subscribe bridge, the link also enables WebSphere Application Server applications to subscribe to messages published by WebSphere MQ applications, and WebSphere MQ applications to subscribe to messages published by WebSphere Application Server applications.

The WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group that provides the gateway for a WebSphere MQ link to connect to a WebSphere MQ network is known as the *gateway queue manager*. When you define a WebSphere MQ link, you nominate one WebSphere MQ queue manager or queue-sharing group to act as a gateway queue manager. This queue manager or queue-sharing group must exist and be active in the WebSphere MQ network

To the gateway queue manager, the messaging engine with the WebSphere MQ link (and hence the bus that the messaging engine is a member of) appears to be a WebSphere MQ queue manager. To the messaging engine with the WebSphere MQ link, the gateway queue manager (and any other queue managers connected to it) appears to be a foreign bus.

The WebSphere MQ link has sender and receiver channels defined on it. These channels communicate with, respectively, partner receiver and sender channels on the gateway queue manager. The WebSphere MQ link communicates with WebSphere MQ using the WebSphere MQ message formats and communication protocol. The WebSphere MQ link converts the service integration message formats to and from the WebSphere MQ message formats.

When WebSphere Application Server applications send messages over the WebSphere MQ link, the messages are transmitted through the WebSphere MQ link sender channel to the partner receiver channel on the gateway queue manager. The receiver puts the messages to the target destinations in the WebSphere MQ network.

Messages from the WebSphere MQ network, that are destined for WebSphere Application Server applications, are sent to the gateway queue manager. The sender channel on the gateway queue manager transmits the messages to the WebSphere MQ link receiver channel, from where they are distributed to the target destinations on WebSphere Application Server.

Network topologies for interoperation using a WebSphere MQ link

These examples show a range of network topologies, from simple to complex, that enable WebSphere Application Server to interoperate with WebSphere MQ using a WebSphere MQ link.

- “Single WebSphere Application Server application server connected to a single WebSphere MQ queue manager”
- “WebSphere Application Server cell connected to a WebSphere MQ network” on page 411
- “High availability for a WebSphere Application Server cell connected to a WebSphere MQ network” on page 412
- “Multiple WebSphere Application Server cells connected to a WebSphere MQ network” on page 414

For completeness, this topic describes a wide range of topologies, including clustered and highly available topologies. Note that, for clustering and high availability, you need to use the network deployment or z/OS version of the product.

Single WebSphere Application Server application server connected to a single WebSphere MQ queue manager

In this basic scenario, a WebSphere MQ link connects a single WebSphere Application Server application server to a WebSphere MQ queue manager. The WebSphere Application Server messaging engine that connects to WebSphere MQ by using the WebSphere MQ link is called the gateway messaging engine. The WebSphere MQ queue manager or queue-sharing group to which the WebSphere MQ link connects is called the gateway queue manager.

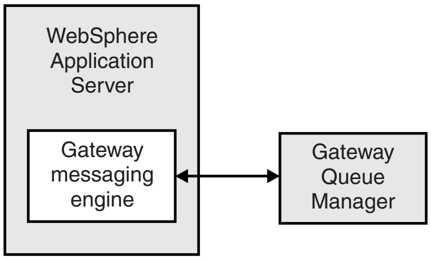


Figure 40. Single application server connected to a gateway queue manager

WebSphere MQ links always use TCP/IP connections, even if the WebSphere MQ queue manager is running on the same host as the application server. You do not need to specify a client or bindings transport type for the connection, as you do when WebSphere MQ is the messaging provider.

The WebSphere MQ link consists of one or two message channels to send messages to WebSphere MQ, receive messages from WebSphere MQ, or both. Each message channel uses one TCP/IP connection.

The message channels support point-to-point messaging between WebSphere Application Server applications and WebSphere MQ applications. You can also configure a publish/subscribe bridge on the WebSphere MQ link for publish/subscribe messaging between WebSphere Application Server applications and WebSphere MQ applications. For more details about the WebSphere MQ link and its message channels, see “Message exchange through a WebSphere MQ link” on page 415.

WebSphere Application Server cell connected to a WebSphere MQ network

A single WebSphere MQ link can connect an entire WebSphere Application Server service integration bus, representing multiple application servers, to multiple WebSphere MQ queue managers. The messages that are exchanged between the two networks all pass through the WebSphere MQ link, which connects a single gateway messaging engine in WebSphere Application Server, and a single gateway queue manager in WebSphere MQ. The gateway messaging engine and gateway queue manager distribute the messages, which can be point-to-point or publish/subscribe messages, to the appropriate application servers and queue managers in their respective networks.

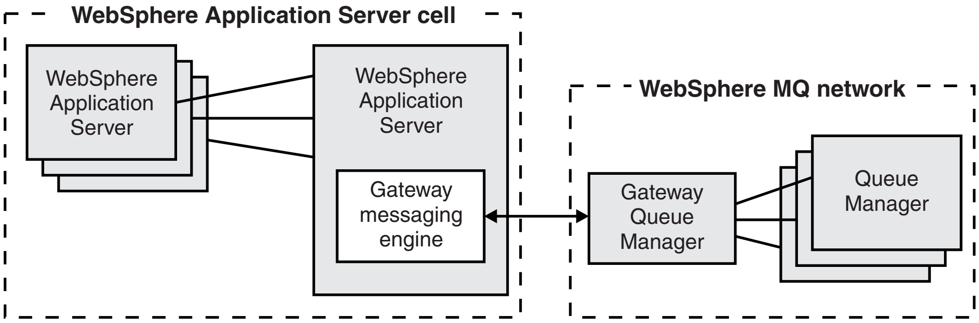

WebSphere Application Server cell
WebSphere Application Server
WebSphere Application Server
Gateway messaging engine
WebSphere MQ network
Gateway Queue Manager
Queue Manager

Figure 41. Multiple application servers connected to multiple queue managers

With this topology, interoperation ceases if any of the following conditions occurs:

- The WebSphere Application Server application server that contains the gateway messaging engine fails.
- The host on which that WebSphere Application Server application server is running fails.
- The WebSphere MQ gateway queue manager fails.
- The host on which the WebSphere MQ gateway queue manager is running fails.

In these situations, none of the application servers in the WebSphere Application Server cell can communicate with any of the queue managers in WebSphere MQ. In the event of a failure, messages are queued as follows:

- If the gateway messaging engine in WebSphere Application Server fails or can no longer communicate with WebSphere MQ, messages that were already queued in the gateway messaging engine, which has store and forward capability, are stored there and are sent when interoperation is restored.
- If the gateway messaging engine in WebSphere Application Server fails, messages that were queued in the messaging engines of other application servers are stored in those messaging engines and are sent when the gateway messaging engine is in operation.
- If the gateway queue manager in WebSphere MQ fails or can no longer communicate with WebSphere Application Server, messages that were already queued in the gateway queue manager are sent when interoperation is restored.
- If the gateway queue manager in WebSphere MQ fails, messages that were queued in other queue managers are sent when the gateway queue manager is in operation.

You can improve the robustness of this topology and introduce greater availability by setting up high availability frameworks in WebSphere Application Server and WebSphere MQ.

High availability for a WebSphere Application Server cell connected to a WebSphere MQ network

The WebSphere Application Server high availability framework eliminates single points of failure and provides peer to peer failover for applications and processes running within WebSphere Application Server. This framework also allows integration of WebSphere Application Server into an environment that uses other high availability frameworks, such as High Availability Cluster Multi-Processing (HACMP), in order to manage non-WebSphere Application Server resources.

Both WebSphere Application Server application servers and WebSphere MQ queue managers can be arranged in clusters, so that if one fails, the others can continue running. In the network topology shown here, the WebSphere Application Server cell that contains the service integration bus now includes a WebSphere Application Server cluster which provides backup for the gateway messaging engine. If the gateway messaging engine fails, it can restart in another application server in the cluster, and it can then restart the WebSphere MQ link to the gateway queue manager. Similarly, the gateway queue manager is part of a WebSphere MQ high-availability cluster.

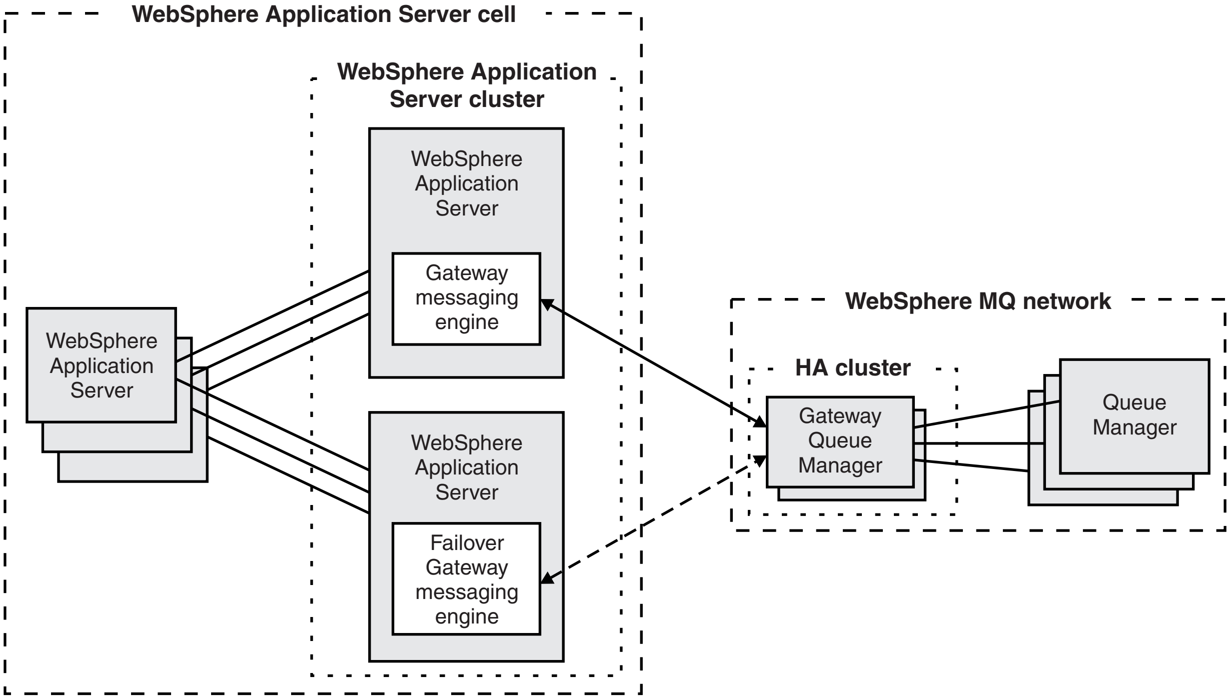

WebSphere Application Server cell
WebSphere Application Server cluster
WebSphere Application Server
Gateway messaging engine
WebSphere Application Server
WebSphere Application Server
Failover Gateway messaging engine
WebSphere MQ network
HA cluster
Gateway Queue Manager
Queue Manager

Figure 42. High availability for multiple application servers connected to multiple queue managers

For WebSphere Application Server and WebSphere MQ to interoperate in this network topology, you must add support for changes of IP address. The WebSphere MQ gateway queue manager uses one IP address to reach the WebSphere Application Server gateway messaging engine, and the WebSphere Application Server gateway messaging engine uses one IP address to reach the WebSphere MQ gateway queue manager. In a high availability configuration, if the gateway messaging engine fails over to a different application server, or the gateway queue manager fails and is replaced by a failover gateway queue manager, the connection to the original IP address for the failed component is lost. You must ensure that both products are able to reinstate their connection to the component in its new location.

To ensure that the connection to a failover WebSphere Application Server gateway messaging engine is reinstated, choose one of the following options:

1. If you are using a version of WebSphere MQ that is earlier than Version 7.0.1, install the SupportPac MR01 for WebSphere MQ. This SupportPac provides the WebSphere MQ queue manager with a list of alternative IP addresses and ports, so that the queue manager can connect with the WebSphere Application Server gateway messaging engine after the messaging engine fails over to a different IP address and port. In WebSphere Application Server you must set a high availability policy of “One of N” for the gateway messaging engine. For more information about the WebSphere MQ MR01 SupportPac, see MR01: Creating a HA Link between WebSphere MQ and a Service Integration Bus.
2. If you are using WebSphere MQ Version 7.0.1, use the connection name (CONNAME) to specify a connection list. Although typically only one machine name is required, you can provide multiple machine names to configure multiple connections with the same properties. The connections are tried in the order in which they are specified in the connection list until a connection is successfully established. If no connection is successful, the channel starts retry processing. When using this option, specify the CONNAME as a comma-separated list of names of machines for the stated TransportType, making sure that all the WebSphere Application Server cluster member IPs are listed directly in the CONNAME. For further information about using the CONNAME, see the WebSphere MQ information center.

Note: WebSphere MQ Version 7.0.1 does not require SupportPac MR01 because this release includes the equivalent function to that provided by SupportPac MR01 for earlier releases. The ability to use the CONNAME to specify a connection list was added as part of the support for multi-instance queue managers in WebSphere MQ Version 7.0.1, however, it can also be used as another option to ensure that the connection to a failover WebSphere Application Server gateway messaging engine is reinstated.

3. Use an external high availability framework, such as HACMP, to manage a resource group that contains the gateway messaging engine. When you use an external high availability framework, the IP address can be failed over to the machine that runs the application server to which the gateway messaging engine has moved. Follow this procedure to handle the IP address correctly:
   - Set a high availability policy of “No operation” for the messaging engine, so that the external high availability framework controls when and where the messaging engine runs.
   - Create resources for the messaging engine and its IP address in the resource group that is managed by the external high availability framework.
   - Consider locating the messaging engine data store in the same resource group as the resource that represents the messaging engine.

To ensure that the connection to a failover WebSphere MQ gateway queue manager is reinstated, choose one of the following options:

1. Set up multi-instance queue managers in WebSphere MQ, as described in the WebSphere MQ information center. In your definition for the WebSphere MQ link sender channel, select **Multiple Connection Names List**, and specify the host names (or IP addresses) and ports for the servers where the active and standby queue managers are located. If the active gateway queue manager fails, the service integration bus uses this information to reconnect to the standby gateway queue manager.
2. Create the WebSphere MQ high-availability cluster using an external high availability framework, such as HACMP, that supports IP address takeover. IP address takeover ensures that the gateway queue manager in its new location appears as the same queue manager to the service integration bus.

The gateway queue manager and the gateway messaging engine store status information that they use to prevent loss or duplication of messages when they restart communication following a failure. This means that the gateway messaging engine must always reconnect to the same gateway queue manager.

If you use WebSphere MQ for z/OS queue sharing groups, you can configure the WebSphere MQ link to use shared channels for the connection. Shared channels provide superior availability compared to the high-availability clustering options available on other WebSphere MQ platforms, because shared channels can reconnect to a different queue manager in the same queue sharing group. Reconnecting in the same queue sharing group is typically faster than waiting to restart the same queue manager in the same or a different location.

Although the network topology described in this section can provide availability and scalability, the relationship between the workload on different queue managers and the WebSphere Application Server application servers to which they are connected is complex. You can contact your IBM representative to obtain expert advice.

Multiple WebSphere Application Server cells connected to a WebSphere MQ network

In this example scenario, a business has two geographically separatedWebSphere Application Server cells, and wants to connect them to the same enterprise-wide WebSphere MQ network. Each service integration bus has its own gateway messaging engine, which connects using a WebSphere MQ link to a nearby WebSphere MQ gateway queue manager.

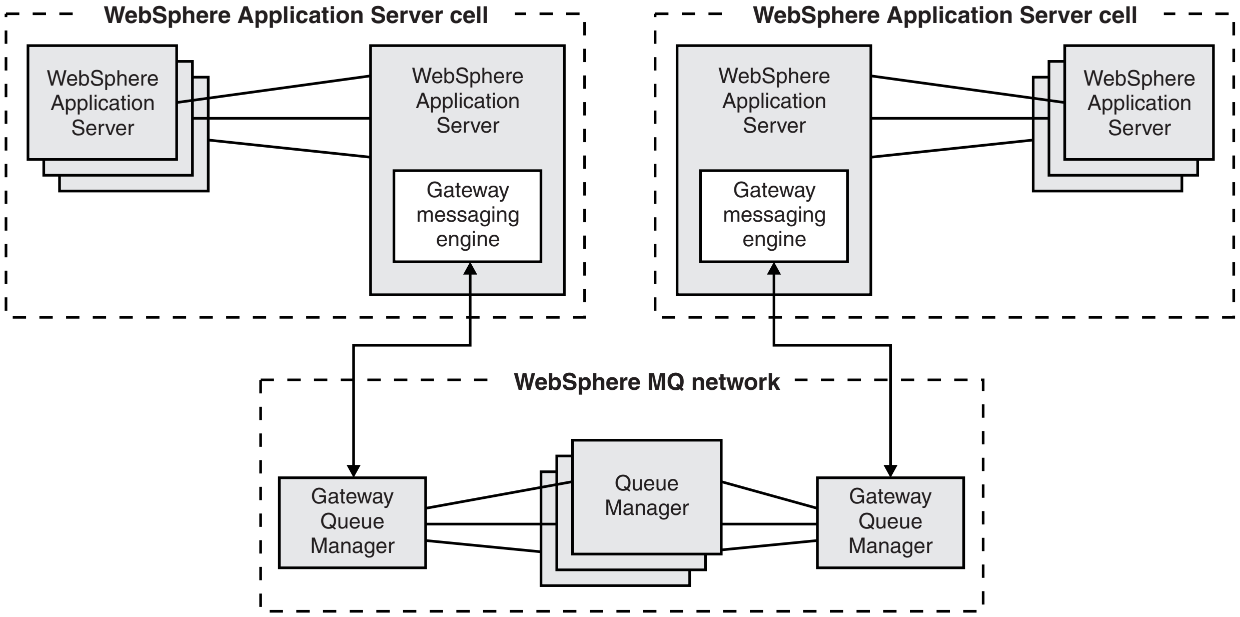

WebSphere Application Server cell
WebSphere Application Server
WebSphere Application Server
Gateway messaging engine
WebSphere Application Server cell
WebSphere Application Server
Gateway messaging engine
WebSphere Application Server
WebSphere MQ network
Gateway Queue Manager
Queue Manager
Gateway Queue Manager

Figure 43. Geographically separated application servers connected to the same WebSphere MQ network

With this network topology, WebSphere Application Server applications running in either WebSphere Application Server cell can exchange point-to-point or (with a publish/subscribe bridge) publish/subscribe messages with WebSphere MQ applications. They can also use the facilities of the enterprise-wide WebSphere MQ network to exchange messages with WebSphere Application Server applications running in the other WebSphere Application Server cell. As in the previous scenario, the business can use high availability frameworks in WebSphere Application Server and WebSphere MQ to provide increased availability and scalability.

Message exchange through a WebSphere MQ link

A WebSphere MQ link connects to a specific foreign bus that represents a WebSphere MQ network, and enables messaging engines on a service integration bus to exchange messages with queue managers on the WebSphere MQ network.

The figure later in this section shows a high level view of the function of a WebSphere MQ link. Subsequent figures add more detail to this simple representation.

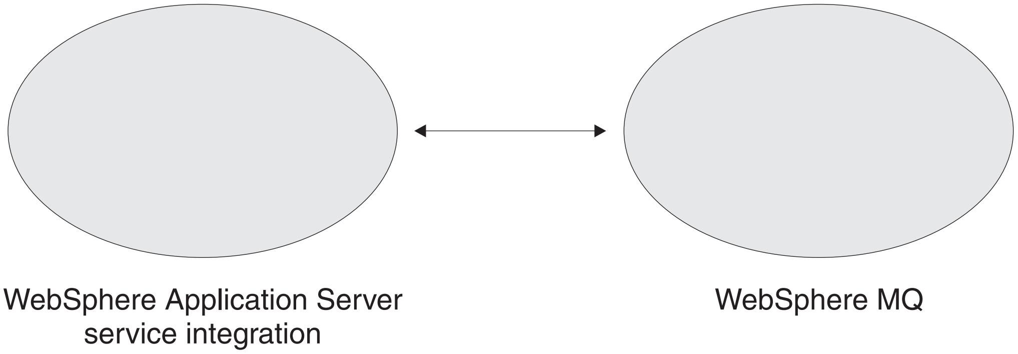


Figure 44. Exchanging messages between WebSphere Application Server and a WebSphere MQ network.

A WebSphere MQ link is a service integration technologies administrative object that describes the attributes required for a messaging engine to establish channel links to a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group.

The messaging engine that connects to WebSphere MQ by using a WebSphere MQ link is known as the *gateway messaging engine*. The WebSphere MQ queue manager or queue-sharing group to which a WebSphere MQ link connects is known as the *gateway queue manager*. To service integration, the gateway queue manager and any other queue managers connected to it appear to be a foreign bus, which is another bus that has a link to the local bus. To the gateway queue manager, the service integration bus appears to be a remote queue manager.

The figure later in this section shows an application server that is a member of a bus and therefore contains a messaging engine. The messaging engine is a gateway messaging engine, which means it connects to a gateway queue manager within WebSphere MQ by using a WebSphere MQ link. The link appears to the gateway queue manager as a message channel - that is, a sender channel, a receiver channel, or a sender-receiver pair of channels.

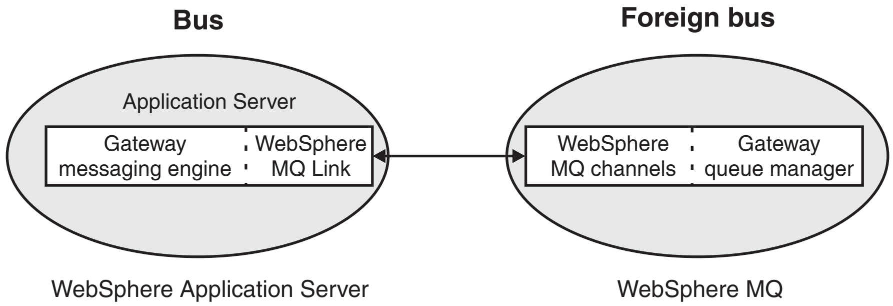


Figure 45. Exchanging messages between a service integration bus, and a foreign bus in a WebSphere MQ network.

Other messaging engines on the same service integration bus can use the gateway messaging engine to send messages to, and receive messages from, the gateway queue manager on WebSphere MQ. In a similar way, the gateway queue manager receives messages from the WebSphere MQ link and routes them to other queue managers in the WebSphere MQ network. The gateway queue manager and the other queue managers to which it connects are together represented as a foreign bus when you configure the WebSphere MQ link.

A WebSphere MQ link cannot use cluster-sender and cluster-receiver channels to connect to multiple queue managers in a WebSphere MQ cluster. Even if the gateway queue manager is a member of a cluster, the WebSphere MQ link must still connect directly to the gateway queue manager. The gateway queue manager manages routing of messages to other queue managers in the cluster.

The figure later in this section shows how messages exchanged between the gateway messaging engine and the gateway queue manager, can be sent and received by other messaging engines on the same bus and other queue managers connected to the gateway queue manager.

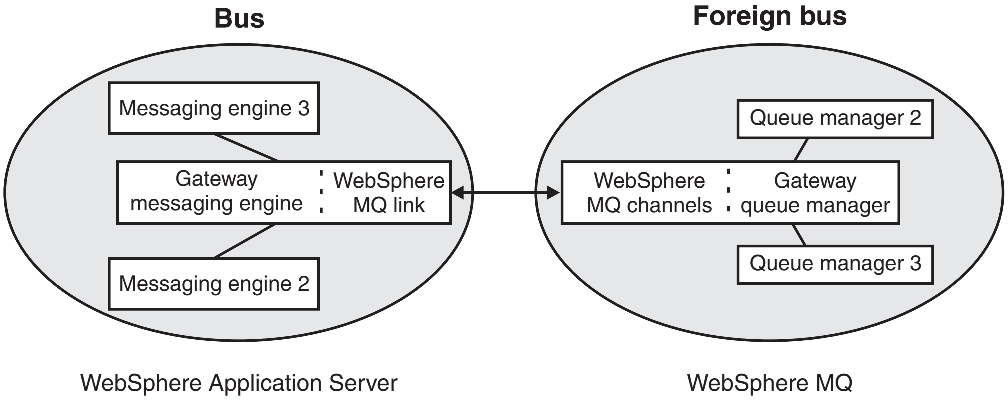


Figure 46. Exchanging messages between messaging engines on a bus and queue managers connected to the gateway queue manager on a foreign bus.

A WebSphere MQ link can have definitions for a WebSphere MQ link sender or a WebSphere MQ link receiver or both. The link sender and receiver emulate the behavior of WebSphere MQ sender and receiver channels. The MQ link sender therefore sends messages to the receiver channel of the gateway queue manager, and the MQ link receiver receives messages from the sender channel of the gateway queue manager.

The figure later in this section shows the sender and receiver channels that enable the gateway messaging engine and the gateway queue manager to exchange messages.

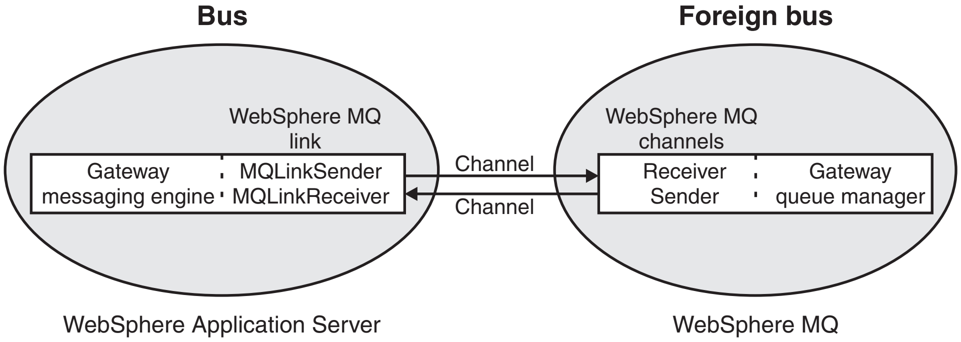


Figure 47. Exchanging messages between WebSphere MQ link sender and receiver channels, and a gateway queue manager with receiver and sender channels.

The figure later in this section shows how an individual message passes from the gateway messaging engine with a WebSphere MQ link, to the target queue in the WebSphere MQ network, and how a response message is returned over the WebSphere MQ link to a reply-to destination in WebSphere Application Server.

1. A service integration JMS application sends a request message to a target destination, which is a JMS destination that points to a WebSphere MQ queue. The sending application includes the reply-to destination in a header field in the request message. The reply-to destination is a JMS destination that points to a service integration destination in the same service integration bus to which the sending application is attached.
2. The messaging engine in the service integration bus uses the WebSphere MQ link to send the message to WebSphere MQ. WebSphere MQ puts the message on the target queue.
3. The WebSphere MQ application receives the message from the queue, processes it, and sends a response to the reply-to destination. This application might be, but is not always, a JMS application.

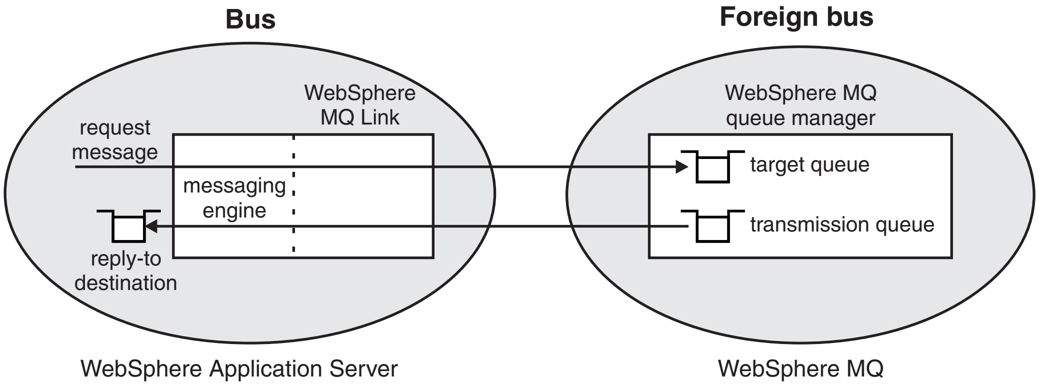


Figure 48. Paths taken by a message and response exchanged between a messaging engine on a bus and a queue manager in WebSphere MQ

You can configure a publish/subscribe bridge on a WebSphere MQ link. The bridge allows subscribing applications connected to the service integration bus to receive messages from publishing applications connected to the WebSphere MQ network. The same publish/subscribe bridge allows subscribing applications connected to the WebSphere MQ network to receive messages from publishing applications connected to the service integration bus.

If you want to specify service integration bus destination attributes for a WebSphere MQ queue, or if you want to control access to a WebSphere MQ queue from service integration bus applications, then you can define a foreign destination to represent the WebSphere MQ queue. If you want your service integration bus applications to use a different name for the WebSphere MQ queue then you can define an alias destination.

The WebSphere MQ link communicates with WebSphere MQ by using WebSphere MQ format and protocols. To identify a supported version of WebSphere MQ, see the supported hardware and software web page at WebSphere Application Server detailed system requirements.

WebSphere MQ link sender:

The WebSphere MQ link sender converts messages to WebSphere MQ format messages, and then sends them to a receiver channel on the WebSphere MQ gateway queue manager or (for WebSphere MQ for z/OS) queue-sharing group.

The WebSphere MQ link sender, which is part of the WebSphere MQ link, emulates the behavior of the sender channel in WebSphere MQ.

- You can define the attributes of the MQ link sender when you define the WebSphere MQ link. The foreign bus connection wizard helps you to do this.

- A number of MQ link sender and MQ link receiver attributes, which are grouped together on the same administrative console panel, are common. That is, the value you enter is used for both the sending and receiving ends of the link.
- If the initial state of the MQ link sender is set to `Started`, the MQ link sender starts when the WebSphere MQ link starts. When the WebSphere MQ link is set to `Stopped`, the MQ link sender stops.
- If an MQ link sender is stopped, the WebSphere MQ link becomes unavailable and messages are held in the service integration bus until the MQ link sender is started again.
- If the MQ link sender encounters a long reply-to destination name (that is, too long a name for WebSphere MQ to handle), it sends the message to an exception destination.
- Define an MQ link sender only if you want to send messages to a WebSphere MQ network. You do not need an MQ link sender if you want only to receive incoming messages.

On the WebSphere MQ link sender channel, there is a field called **`Transport chain`**. This field can take a value of `OutboundBasicMQLink`, which communicates with an unsecured channel, or `OutboundSecureMQLink`, which communicates with a secure SSL channel. The unsecured listener listens for inbound requests on port 5558 and the secure listener listens for inbound requests on port 5578.

WebSphere MQ link receiver:

The WebSphere MQ link receiver receives messages sent to a messaging engine over a WebSphere MQ link. The messages are sent from a sender channel on a WebSphere MQ gateway queue manager or (for WebSphere MQ for z/OS) queue-sharing group in a WebSphere MQ network to a WebSphere MQ link on a messaging engine.

The WebSphere MQ link receiver, which is part of the WebSphere MQ link, emulates the behavior of the receiver channel in WebSphere MQ.

- You can define the attributes of the MQ link receiver when you define the WebSphere MQ link. The foreign bus connection wizard helps you to do this.
- A number of MQ link sender and MQ link receiver attributes, which are grouped together on the same administrative console panel, are common. That is, the value you enter is used for both the sending and receiving ends of the link.
- The MQ link receiver communicates with a WebSphere MQ sender channel on the gateway queue manager or queue-sharing group, and converts messages in WebSphere MQ format to messages in service integration format.
- If the initial state of the MQ link receiver is set to `Started`, the MQ link receiver starts when the WebSphere MQ link starts, which means it is available when senders connect to it. When the WebSphere MQ link is set to `Stopped`, the MQ link receiver stops.
- If an MQ link receiver is stopped, the WebSphere MQ link becomes unavailable and messages are held on the transmission queue on the WebSphere MQ gateway queue manager or queue-sharing group in the WebSphere MQ network.
- Define an MQ link receiver only if you want to receive messages from a WebSphere MQ network. You do not need an MQ link receiver if you want only to send outgoing messages.

Point-to-point messaging with a WebSphere MQ network

The WebSphere MQ link, defined on a messaging engine in the service integration bus, describes the attributes required to connect to, and send or receive messages to or from, a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue sharing group that acts as a gateway to the WebSphere MQ network.

Point-to-point messaging might be:

- A request from WebSphere Application Server to WebSphere MQ, optionally followed by a WebSphere MQ reply.
- A request from a WebSphere MQ network, optionally followed by a WebSphere Application Server reply.

The following figure shows the flow of point-to-point messages across the WebSphere MQ link.

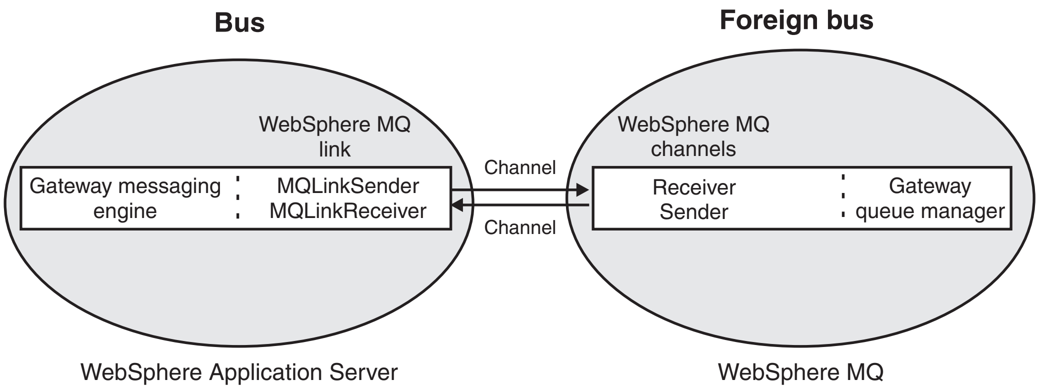


Figure 49. Exchanging messages between WebSphere MQ link sender and receiver channels, and a gateway queue manager with receiver and sender channels.

See “Request-reply messaging through a WebSphere MQ link” on page 426 for more information about the reply messages transmitted across the WebSphere MQ link.

Point-to-point messaging might also include:

- A request from WebSphere Application Server through a WebSphere MQ network to another WebSphere Application Server, and a reply from that WebSphere Application Server, again through WebSphere MQ. For details of this two-stage messaging flow model, see “Messaging between two application servers through WebSphere MQ” on page 430.
- A request from a WebSphere MQ network through a WebSphere Application Server to another WebSphere MQ network, and a reply from that WebSphere MQ network, again through a WebSphere Application Server. For details of this two-stage messaging flow model, see “Messaging between two WebSphere MQ networks through an application server” on page 431.

The following figure shows how messages can be exchanged between applications and messaging engines that are on the same bus, as well as between the WebSphere MQ link and queue managers connected to the gateway queue manager in the WebSphere MQ network.

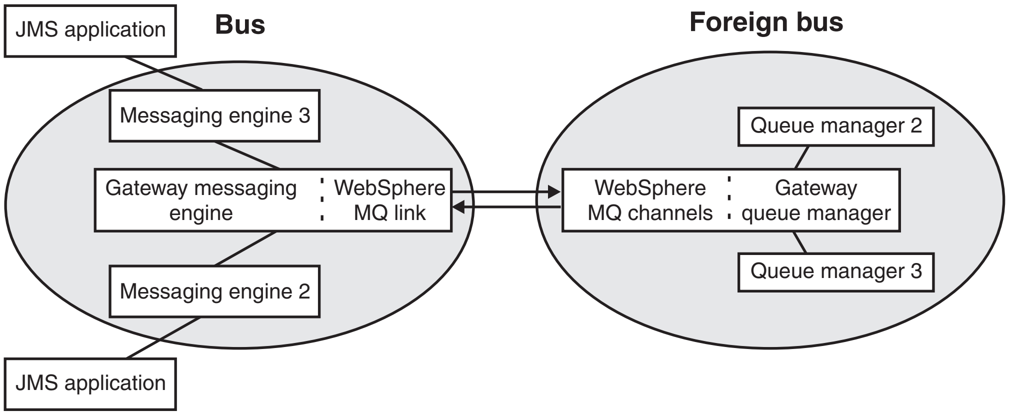


Figure 50. Exchanging messages between messaging engines on a bus that has a WebSphere MQ link that is connected to a gateway queue manager on a foreign bus.

Notes:

- If your WebSphere Application Server application sends point-to-point messages to a WebSphere MQ application that is not JMS, such as a WebSphere MQ message-driven application in CICS (using the CICS MQ bridge) or IMS (using the IMS MQ bridge), then your WebSphere Application Server application has to use special techniques to ensure that the service integration messages (most likely JMS messages) are presented to the non-JMS application in a way that the application can understand. For more information, see Programming for interoperation with WebSphere MQ , "How service integration converts messages to and from WebSphere MQ format" on page 405, and How to process WebSphere MQ message headers, which describes WebSphere Application Server helper classes that assist in the creation of suitable headers and body content.
- Some WebSphere MQ applications can process messages that include an MQRFH2 header (generally these are JMS or XMS or WebSphere MQ Version 7 applications) and some applications cannot do so (generally these are WebSphere MQ applications that predate the MQRFH2 header). You must set the destination context to inhibit adding an MQRFH2 header when messages are destined for a WebSphere MQ application that cannot handle this header. For information about setting the destination context, see Specifying whether messages are forwarded to WebSphere MQ as JMS messages. The MQRFH2 header contains fields unique to the service integration bus. For details of these fields, see Mapping additional MQRFH2 header fields in service integration.
- Any WebSphere MQ queue name is also valid as a bus destination name and, as a general rule, you should configure a bus destination that is a WebSphere MQ queue to use the WebSphere MQ queue name. If your bus applications have to use a different name, you can achieve this by using an alias destination.
- WebSphere MQ channel or conversion exits (for example, for data conversion) are not supported by the WebSphere MQ link.

Publish/subscribe messaging through a WebSphere MQ link

On a WebSphere MQ link, you can set up publish/subscribe messaging between WebSphere Application Server and the WebSphere MQ publish/subscribe function, or WebSphere Message Broker, or WebSphere Event Broker.

The following product versions provide publish/subscribe capability that you can use with WebSphere Application Server over a WebSphere MQ link:

WebSphere MQ Version 7
Provides publish/subscribe function that is integrated into WebSphere MQ queue managers. This capability is called integrated publish/subscribe. The publish/subscribe capability for earlier versions of WebSphere MQ is called queued publish/subscribe, because you communicate with a separate publish/subscribe broker by means of messages placed on queues.

WebSphere Message Broker Version 6
Provides a separate publish/subscribe broker for queued publish/subscribe. This version of queued publish/subscribe uses MQRFH2 message headers. If you are using WebSphere MQ Version 6, you can use WebSphere Message Broker Version 6 to provide publish/subscribe function that interoperates over a WebSphere MQ link with WebSphere Application Server. From WebSphere Message Broker Version 7, this product no longer provides a separate publish/subscribe broker, and all topic-based publish/subscribe operations made through the product use WebSphere MQ facilities.

WebSphere Event Broker
Provides a separate publish/subscribe broker for queued publish/subscribe. This version of queued publish/subscribe is the same as that in WebSphere Message Broker Version 6.

WebSphere MQ Version 6 (except Version 6 of WebSphere MQ for z/OS) provides queued publish/subscribe that is implemented by a publish/subscribe broker within WebSphere MQ. However, the publish/subscribe function provided by WebSphere MQ Version 6 uses MQRFH message headers, also known as MQRFH1 message headers, and it does not support the MQRFH2 message headers that the WebSphere Application Server publish/subscribe bridge uses on the WebSphere MQ link. For WebSphere MQ Version 6 publish/subscribe to interoperate with WebSphere Application Server publish/subscribe over a WebSphere MQ link, you must use a separate message broker product that supports MQRFH2 message headers. Alternatively, instead of using a WebSphere MQ link, you can interoperate using the WebSphere MQ messaging provider, so that your applications use WebSphere MQ publish/subscribe function and do not use the service integration bus. This requirement also applies to earlier versions of WebSphere MQ where publish/subscribe function is provided by SupportPac MA0C.

Message headers and contents are mapped in the same way for both point-to-point messages and publish/subscribe messages. For more information about the mapping of messages see “How service integration converts messages to and from WebSphere MQ format” on page 405.

Publish/subscribe bridge on a WebSphere MQ link:

A publish/subscribe bridge enables publish/subscribe messaging between WebSphere Application Server and WebSphere MQ through a WebSphere MQ link. The publish/subscribe bridge provides a connection between the publish/subscribe function of a service integration bus and the publish/subscribe function of a WebSphere MQ network.

When you use WebSphere MQ integrated publish/subscribe, the publish/subscribe bridge can connect as a subscriber or publisher to queue managers in the WebSphere MQ network:

- To act as a subscriber, the publish/subscribe bridge connects to a WebSphere MQ queue manager with a durable subscription to the relevant topic, and so receives messages when they are published on that topic. The publish/subscribe bridge then passes the messages to subscribers connected to a service integration bus in WebSphere Application Server. These subscribers might be applications running in WebSphere Application Server, or they might be bus clients running in Java Platform, Standard Edition (Java SE) or third party application servers.
- To act as a publisher, the publish/subscribe bridge subscribes to messages that applications have published to a service integration JMS topic destination in WebSphere Application Server. The publish/subscribe bridge then publishes the messages on the relevant topic in WebSphere MQ, and the WebSphere MQ queue manager distributes the messages to the subscribers in the WebSphere MQ network. Service integration can also send the messages to other subscribers that are connected to the service integration bus in WebSphere Application Server.

The publish/subscribe bridge acts in the same way if you use a queued publish/subscribe capability provided by a compatibility interface within WebSphere MQ Version 7, or by a separate message broker product. The publish/subscribe bridge attaches to the publish/subscribe broker as either a subscriber or a publisher, and receives messages from the message broker or publishes them to the message broker. The message broker distributes published messages to its subscribers in the WebSphere MQ network.

If communication between the two ends of the publish/subscribe bridge stops, messages are held until communication is reestablished by the system or by the administrator. If you are using a separate message broker product, the messages might be held on the input queues for the broker, if the broker is not available, or on the transmission queue for WebSphere MQ, if WebSphere MQ is not available.

The publish/subscribe bridge consists of the broker profiles and topic mappings that you have defined on the WebSphere MQ link:

- A broker profile defines a connection to a single WebSphere MQ queue manager or separate publish/subscribe broker. For more information about broker profiles, see “Broker profile on a WebSphere MQ link” on page 423.
- A topic mapping defines how messages on a particular topic flow between the two ends of the publish/subscribe bridge. For more information about topic mappings, see “Topic mapping on a WebSphere MQ link” on page 423.

Broker profile on a WebSphere MQ link:

A broker profile on a WebSphere MQ link defines a connection through a WebSphere MQ link to a WebSphere MQ queue manager, for the purpose of publish/subscribe messaging with a WebSphere MQ network.

A broker profile applies to the connection between WebSphere Application Server and a single WebSphere MQ queue manager or separate publish/subscribe broker. It contains the following information:

- The name of the WebSphere MQ link.
- The name of the service integration bus in WebSphere Application Server.
- The name of the messaging engine in WebSphere Application Server.
- The name of the queue manager in WebSphere MQ. This queue manager does not have to be the same as the WebSphere MQ gateway queue manager, provided that it can be reached from the WebSphere MQ gateway queue manager. If you are using a separate publish/subscribe broker, this queue manager is the WebSphere MQ queue manager to which the message broker is connected. You must ensure that the service integration bus has sufficient authority to send subscription requests to the queue manager or separate publish/subscribe broker.

You can define multiple broker profiles on a single WebSphere MQ link, to connect to multiple queue managers in the WebSphere MQ network.

You define one or more topic mappings for a broker profile. A topic mapping links a specific topic in the WebSphere Application Server service integration bus with its equivalent in the WebSphere MQ network. For more information about topic mappings, see “Topic mapping on a WebSphere MQ link.”

The broker profiles, along with their topic mappings, form a publish/subscribe bridge with the WebSphere MQ network. The publish/subscribe bridge connects as a subscriber to receive messages from the WebSphere MQ network and pass them to applications in WebSphere Application Server, and it connects as a publisher to publish messages on topics in the WebSphere MQ network. For a description of how the publish/subscribe bridge operates, see “Publish/subscribe bridge on a WebSphere MQ link” on page 422.

Topic mapping on a WebSphere MQ link:

A topic mapping is an association that defines which messages published in WebSphere Application Server or the WebSphere MQ network, should be forwarded to the other publish/subscribe system.

You define one or more topic mappings for a broker profile. The broker profile defines the connection between WebSphere Application Server and a WebSphere MQ queue manager or separate publish/subscribe broker. The topic mapping links a specific topic in the WebSphere Application Server service integration bus with its equivalent in the WebSphere MQ network.

When you define a topic mapping, you choose if messages are to flow from WebSphere MQ to WebSphere Application Server, or from WebSphere Application Server to WebSphere MQ, or if the flow is two-way, or bidirectional. If a topic mapping is bidirectional, a message is safeguarded from being continually republished on alternating sides of the publish/subscribe bridge.

In your topic mapping, the topic name and its position in the hierarchy (or tree) of topics must be the same in WebSphere Application Server and in the WebSphere MQ network. You can use wild cards, as described in “Wild cards in topic mapping” on page 424. For example, if you set up a topic mapping for "stock/IBM" with a direction from the WebSphere Application Server service integration bus to WebSphere MQ, the publish/subscribe bridge subscribes to the topic "stock/IBM" in WebSphere Application Server, and receives the messages published to the topic. The publish/subscribe bridge then publishes the messages to the topic "stock/IBM" in the WebSphere MQ network.

Messages published by a service integration JMS client are transferred to a WebSphere MQ network if an appropriate topic mapping has been created. This is presented to the WebSphere MQ network as only the topic name, for example, "sports/football". A suitably configured WebSphere MQ JMS application can use this information to publish further information to the same topic, but the original WebSphere Application Server JMS application receives these messages only if appropriate topic mapping has been configured.

If you delete a WebSphere MQ link, you must first unsubscribe and delete your topic mappings, to ensure there are no outstanding subscriptions in WebSphere Application Server or the WebSphere MQ network. For the process to unsubscribe, see Preparing to remove a foreign bus connection between a service integration bus and a WebSphere MQ network.

Wild cards in topic mapping:

Wild cards can be used in topic mapping on a WebSphere MQ link for publish/subscribe messaging, but there are differences between wild cards in WebSphere Application Server service integration, and wild cards in a WebSphere MQ network.

Publish/subscribe messaging in the WebSphere MQ network uses wild cards # and +. These wild cards are represented as /. and * respectively in the service integration bus. However, the publish/subscribe bridge on the WebSphere MQ link supports only the /. wild card, and only at the end of the topic. For example, stock//. in the service integration bus is equivalent to stock/# in the WebSphere MQ network, meaning all messages with "stock" at the beginning of the topic.

When you use wild cards in the topic mappings that you enter on the WebSphere MQ link administrative console panels, you must use the publish/subscribe bridge symbols, not the equivalent WebSphere MQ network symbols. The publish/subscribe bridge handles conversion automatically.

Publish/subscribe messaging through a WebSphere MQ link: example:

A publish/subscribe bridge over a WebSphere MQ link enables subscribers on the service integration bus in WebSphere Application Server to receive the same published messages as subscribers in a WebSphere MQ network. The broker profile in WebSphere Application Server allows these two separate publish/subscribe domains to appear as a single entity.

Imagine that there are two businesses “GolfStats Inc.” and “FootballFansData Inc.” that each provide a results and news service for different types of sporting event. Both pay third parties to collect sports information (for golf and football respectively) and publish this data to their IT systems. GolfStats and FootballFansData then charge members of the public a monthly fee in exchange for an application that runs on a desktop computer, which pops up the results as they become available.

GolfStats also use their IT system to host a website and run other business applications, so their IT systems are based on WebSphere Application Server and the service integration bus. However, FootballFansData do not have any other business applications, and they use WebSphere MQ messaging for their publish/subscribe requirements.

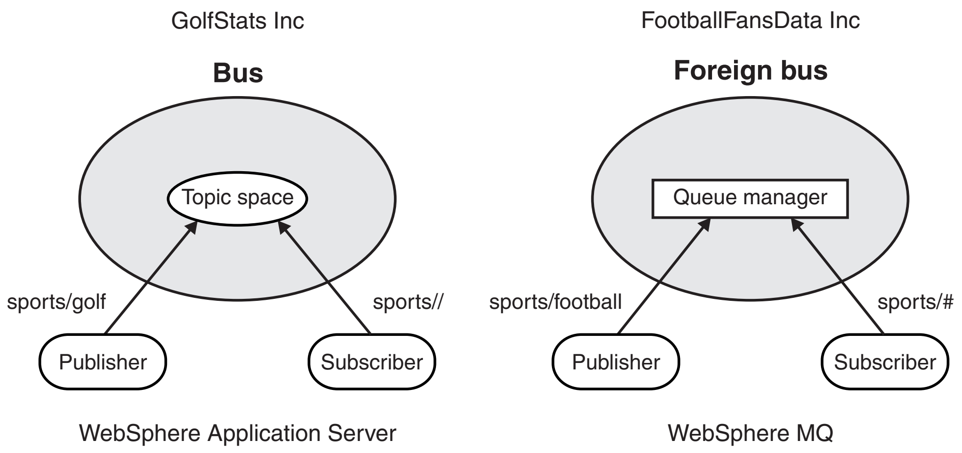


Figure 51. Two separate businesses that publish information to their respective audiences.

Figure 1 shows two separate businesses. GolfStats Inc has a third party that connects to their IT systems when a result becomes available and publishes information to a topic space on the topic “sports/golf”, which is received by the subscribers subscribing to “sports//.”. (//. in the syntax used by the publish/subscribe bridge indicates all sports information). Publish/subscribe messaging in GolfStats Inc is handled by a service integration bus.

Similarly a third party supplier for FootballFansData Inc publishes information to the WebSphere MQ network on the topic “sports/football”, which is received by a subscriber application subscribing to “sports/#” (WebSphere MQ syntax for all sports information). Publish/subscribe messaging in FootballFansData Inc is handled by a WebSphere MQ queue manager, which would be viewed by WebSphere Application Server as a foreign bus, although the two systems are not currently connected.

Recently GolfStats and FootballFansData have merged, and the new management want to join the existing IT systems together in order to provide information about golf and football to both sets of customers. One option is to migrate all FootballFansData's IT systems to use the service integration bus. However, this approach requires significant capital investment, as well as upgrading the third party and customer application code to be able to connect to the system. A simpler alternative is to bridge between the two systems by using the WebSphere MQ link and a broker profile.

The businesses take the following actions to bridge between the two systems:

1. Identify a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group, named (for example) QM_GATEWAY on the FootballFansData system, that will act as the gateway to connect to the WebSphere MQ network.
2. Configure a Foreign bus connection for the GolfStats service integration bus to enable messages to be exchanged between the bus and the WebSphere MQ network.
3. Define a broker profile on the WebSphere MQ link that states the name of the queue manager in the WebSphere MQ network where the messages are published, named QM_TWO in this example.
4. Define a topic mapping associated with the broker profile to allow publications to flow between the service integration bus and the WebSphere MQ network. The mapping will be bidirectional on a topic of “sports//.”, which allows all publications in the sports branch of the topic hierarchy to be transferred.

When these tasks have been completed, and the application server that hosts the GolfStats service integration bus has been restarted, messages begin to flow between the two systems. This enables the FootballFansData customers to receive information about golf, and the GolfStats customers to receive information about football. The diagram later in this section shows the logical path of a “golf” message published into the GolfStats IT system being received by a subscriber on the FootballFansData system.

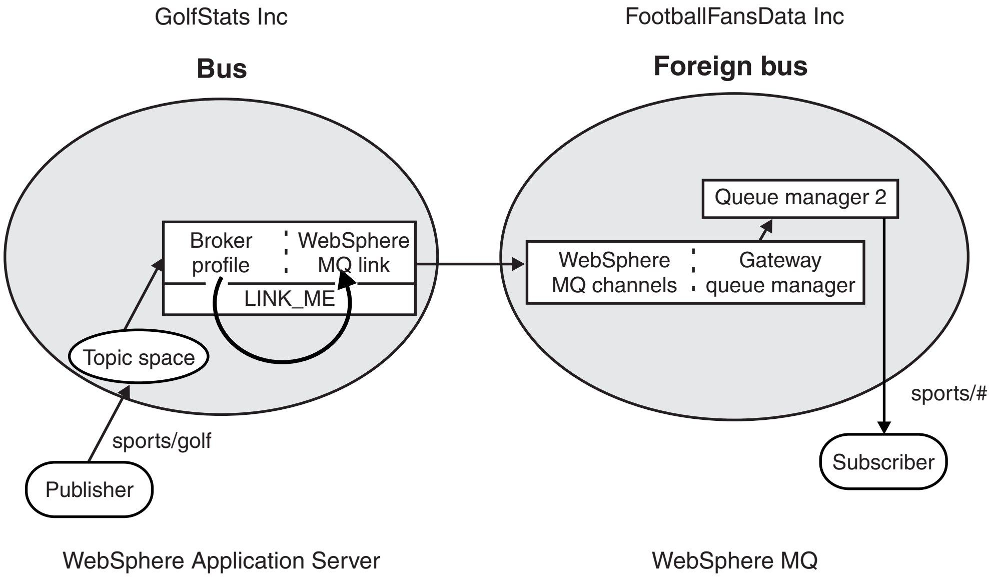


Figure 52. Two linked businesses with one of them publishing to the other.

If GolfStats used the same topic space to publish information on the topic “business/financials” for internal consumption by staff, then these messages would not be routed to the WebSphere MQ network of FootballFansData because a topic mapping has not been created for this topic. This ensures that the GolfStats team can limit the people who are able to receive these messages to people authorized to do so on the GolfStats system, and avoid unnecessary message traffic between the two systems.

Request-reply messaging through a WebSphere MQ link

When a JMS producer sends a message, it can provide a reply-to destination. The reply-to destination is a JMS destination defined using the producer's messaging provider. This style of messaging is known as request-reply, or request and response. Request-reply messages can be exchanged across the WebSphere MQ link as either point-to-point or publish/subscribe messages.

The following combinations are possible for request-reply exchanges:

1. Publication messages (for publish/subscribe messaging).
   a. Queue type reply-to destination.
   b. Topic type reply-to destination. This can be a permanent reply-to topic, which is defined to the publish/subscribe bridge on the WebSphere MQ link, or a temporary topic reply, where the topic name is assigned at runtime.
2. Point-to-point messages.
   a. Queue type reply-to destination.
   b. Topic type reply-to destination. The reply-to destination must be a permanent reply-to topic; you cannot have a temporary topic reply for a point-to-point request message.

When the producer is a WebSphere Application Server application that uses the default messaging provider, the reply-to destination is a service integration JMS destination that typically either points to a service integration queue, or comprises a topic string and a service integration topic space.

- If the reply-to destination is a service integration queue, it is normally a queue in the same bus as the WebSphere Application Server application, so that the WebSphere Application Server application can consume the reply message from that reply-to queue.

- If the reply-to destination is a topic and the consumer is a WebSphere MQ program, you must configure the publish/subscribe bridge to ensure that the reply message can be routed back to the service integration bus so that the WebSphere Application Server application can receive it. Note that although WebSphere MQ JMS applications can reply to a topic, most other WebSphere MQ applications cannot.

When the producer is a WebSphere MQ JMS application, the reply-to destination is a WebSphere MQ JMS destination that typically either points to a WebSphere MQ queue or is a topic string.

- If the reply-to destination is a WebSphere MQ queue, it is normally a queue in the queue manager, or shared queue in the queue-sharing group, that the WebSphere MQ application is using, so that the WebSphere MQ application can consume the reply message from that reply-to queue.
- If the reply-to destination is a topic and the consumer is a WebSphere Application Server application, you must configure the publish/subscribe bridge to ensure that the reply message can be routed back to WebSphere MQ so that the WebSphere MQ application can receive it.

Reply-to queues for request-reply messaging through a WebSphere MQ link:

Reply-to queues indicate to a receiving application where a reply should be sent. You can use reply-to queues for point-to-point request messages (queues) and for publish/subscribe request messages.

The reply-to queue might be a predefined queue or a dynamically created temporary or permanent queue. If it is a dynamic queue, then it might have a unique name that is generated by WebSphere MQ. WebSphere Application Server messaging technology has the same concept of a temporary queue for replies, and generates a queue name of up to 48 characters to comply with the queue-name length limitation for WebSphere MQ.

Deciding to use reply-to queues is part of application design (see Designing an application for interoperation with WebSphere MQ). Your sending application must contain a definition of where replies are to be sent and attach this information to its messages. The replying application uses this data in the received message to discover the name of the queue to which to reply.

There are two fields in the JMS API that are used for sharing information about the destination to which a message is sent (**JMSDestination**) and the destination to which replies should be sent (**JMSReplyTo**). The **JMSReplyTo** field allows a response message to be returned if required. It contains enough detail for the receiving application to send a response message to the intended queue or topic so that it can be read by an application associated with the sender of the request. The **JMSReplyTo** field of a JMS message passing from a service integration bus to WebSphere MQ (or from WebSphere MQ to a service integration bus) is automatically mapped so that a consuming application in WebSphere MQ can reply to the original WebSphere Application Server application.

Reply-to topics for request-reply messaging through a WebSphere MQ link:

WebSphere Application Server and WebSphere MQ JMS applications can publish messages to a topic space with a reply-to topic. Applications in the other network can receive the message, obtain the reply destination, and publish a message on the reply topic. Topic type replies cross the WebSphere MQ link through the publish/subscribe bridge.

Topic type replies are of two varieties:

1. Permanent reply-to topics.
2. Temporary topic replies.

Topic type reply-to destinations cannot be used with most WebSphere MQ applications. WebSphere MQ JMS applications handle them correctly, but MQI applications do not.

Permanent reply-to topics

For a reply message published to a permanent topic to cross between WebSphere Application Server and a WebSphere MQ network, the administrator must define an appropriate topic mapping for the reply topic on the publish/subscribe bridge for the WebSphere MQ link. The topic mapping defines the topic name and specifies whether messages are to flow from WebSphere MQ to WebSphere Application Server, or from WebSphere Application Server to WebSphere MQ, or if the flow is two-way, or bidirectional.

For example, a WebSphere MQ JMS application is publishing messages on the topic "myTopic" in the WebSphere MQ network. The messages have a reply topic of "myReplyTopic". A WebSphere Application Server JMS application needs to receive the messages and publish replies to the reply topic. For this exchange of messages, you must specify two topic mappings on the publish/subscribe bridge:

- A topic mapping to make the publish/subscribe bridge subscribe to "myTopic" in the WebSphere MQ network. With this topic mapping, when the WebSphere MQ JMS application publishes messages to "myTopic", the messages are sent over the WebSphere MQ link, translated into the correct format, and delivered to the publish/subscribe bridge subscriber queue. There, they are processed and then sent on to the topic space as specified in the publish/subscribe topic mapping. The WebSphere Application Server JMS application receives the messages from the topic space.
- A topic mapping to make the publish/subscribe bridge forward messages published to "myReplyTopic" in WebSphere Application Server to the WebSphere MQ network. With this topic mapping, when the WebSphere Application Server JMS application publishes reply messages to "myReplyTopic" in WebSphere Application Server, the publish/subscribe bridge sends them to the WebSphere MQ network, where they are also published to "myReplyTopic".

Temporary topic replies

For a temporary topic reply message to cross between WebSphere Application Server and a WebSphere MQ network, you do not define a separate topic mapping. A temporary topic name is assigned at runtime, and the reply message is automatically routed between WebSphere Application Server and a WebSphere MQ network by the publish/subscribe bridge.

However, for temporary topic reply messages to be routed from the service integration bus back to WebSphere MQ through the publish/subscribe bridge, you must configure the broker stream queue of the topic mapping on which the request message is sent. The broker stream queue is the queue where the messages are published. This field will already be specified for bi-directional topic mappings. Although this field is not mandatory for "From MQ" topic mappings, it must be completed if you want temporary topic reply messages to be routed.

Temporary topic replies are only supported to publication messages. Point-to-point request messages with temporary topic reply destinations are not supported.

Strict message ordering using the strict message ordering facility of the WebSphere Application Server default messaging provider

Strict message ordering can be achieved when deploying message driven bean applications to the WebSphere MQ messaging provider for WebSphere Application Server when no special facilities have been coded into the application to handle messages arriving out of order by using the strict message ordering facility of the WebSphere Application Server default messaging provider.

The following assumptions have been made in this scenario:

- The message-driven bean (MDB) application is transactional.
- The back-out threshold (**BOTHRESH**) on theWebSphere MQ queue has been set to `0`.

Configuration for ordered delivery

- A service integration bus, with a WebSphere MQ link between the WebSphere MQ queue manager hosting the queue and the bus.

- If a mixture of persistent and nonpersistent messages might be sent within an ordered sequence, you must set the non-persistent message speed (**NPMSPEED**) on the WebSphere MQ sender channel to `NORMAL`.
- You must configure a destination in the bus with the **Strict message ordering** option selected, which the MDB application consumes from through a default messaging provider activation specification.
- You must replace the local queue definition with a remote queue definition within WebSphere MQ, so that messages that are sent to the destination queue are forwarded over the WebSphere MQ link to the bus.

  Note: This configuration is just one possible option for configuring queue name resolution within the queue manager to forward messages over the link.

Important information about this configuration

- This deployment option combines the message ordering capabilities of WebSphere MQ (which include when sending over a channel) with the additional messages ordering facilities provided by the default messaging provider for WebSphere Application Server (which prevent out of order delivery in transaction recovery scenarios).
- This deployment option is complex as it requires planning, and runtime administration, of a bus topology in addition to a WebSphere MQ topology.
- It also adds internal complexity as messages are converted automatically between the low level WebSphere MQ and default messaging provider formats as they travel over the WebSphere MQ link.

Circumstances in which messages can arrive out of order

There are no circumstances in which this deployment is expected to cause messages to be delivered out of order.

Considerations for clustered deployment

- Ordered delivery from the bus destination to the MDB is enforced automatically in a clustered environment when the **Strict message ordering** option is selected for the destination.
- The main consideration for a clustered environment is establishing high availability of the WebSphere MQ link between the queue manager and the bus. For more information about the options available for ensuring that the connection to a failover WebSphere Application Server gateway messaging engine is reinstated, see .

Securing connections to a WebSphere MQ network

Connections between a WebSphere Application Server and a WebSphere MQ network can use the Secure Sockets Layer (SSL) protocol to increase the confidentiality and integrity of messages transferred between a messaging engine on a service integration bus and WebSphere MQ.

By default, new application servers are configured to accept inbound WebSphere MQ connections through two inbound transport chains. To read about inbound transport chains, see “Inbound transport options” on page 542. One of these chains is configured to accept SSL-based connections, making it possible to configure a sender channel in the WebSphere MQ network to connect through this channel chain and establish an SSL-based connection. For more information about securing WebSphere MQ sender channels, see the *Security* section of the WebSphere MQ information center. All WebSphere MQ interoperation resources hosted by an application server can be contacted by all inbound WebSphere MQ transports defined to that server, so you should restrict the inbound transports that are enabled. This is important because the default application server configuration has definitions for inbound WebSphere MQ transports that are not secured using SSL. For more information, see “Secure transport configuration requirements” on page 546).

When connecting a WebSphere Application Server to a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue sharing group through a WebSphere MQ link sender channel definition, you might choose to secure the link through SSL. This is achieved by specifying a suitable transport chain for the **Transport chain** property of the WebSphere MQ link sender channel definition. The name of the default SSL-based outbound transport chain suitable for securing a WebSphere MQ link sender channel is `OutboundSecureMQLink`. For more information, see "Outbound transport options" on page 544.

Messaging between two application servers through WebSphere MQ

You can use WebSphere MQ links to send a WebSphere Application Server message from one application server to another through a WebSphere MQ network.

You can exchange messages between two application servers through an intermediate WebSphere MQ network, as shown in the following figure:

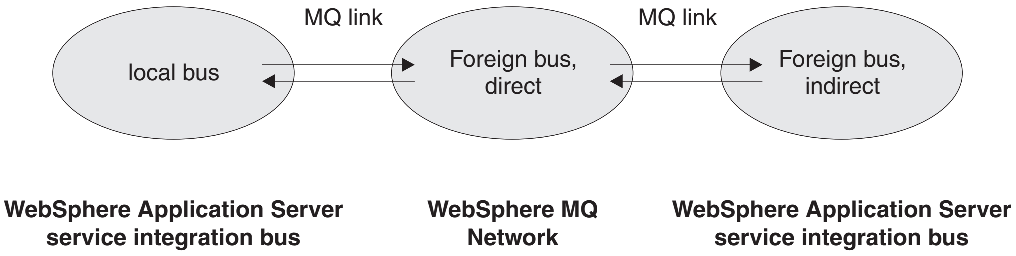

MQ link
MQ link
local bus
Foreign bus, direct
Foreign bus, indirect
WebSphere Application Server service integration bus
WebSphere MQ Network
WebSphere Application Server service integration bus

Figure 53. Exchanging messages between two application servers through an intermediate WebSphere MQ network.

In this case, the WebSphere MQ network includes two gateway queue managers. One connects to the local bus by using a WebSphere MQ sender-receiver pair of message channels, known to the local bus as a WebSphere MQ link. The other connects to the indirect foreign bus by using another WebSphere MQ sender-receiver pair of message channels, known to the indirect foreign bus as a WebSphere MQ link. In the simplest case, the same gateway queue manager connects to both the local bus and the indirect foreign bus.

The WebSphere MQ network must be configured to route messages as required between the local bus and the indirect foreign bus. Details of this configuration are not normally important to WebSphere Application Server administrators, but can be found in WebSphere MQ Intercommunication.

Configuration and operation of messaging between two service integration buses through an intermediate WebSphere MQ network is much more straightforward if you choose bus names that comply with WebSphere MQ queue manager naming restrictions, and if you choose bus destination names that comply with WebSphere MQ queue naming restrictions:

- Queue managers in the WebSphere MQ network "see" the local bus and the indirect foreign bus as queue managers, and refer to them by their virtual queue manager names. If the service integration bus names comply with WebSphere MQ restrictions for queue manager names, the virtual queue manager name that WebSphere MQ uses can (and should) be the same as the bus name used by service integration.

  If the virtual queue manager name that WebSphere MQ uses for a foreign bus is not the same as the service integration bus name used by that foreign bus, the local bus must define the foreign bus by the virtual queue manager name of that foreign bus, not the actual service integration bus name (because the intermediate WebSphere MQ network does not know the actual service integration bus name and cannot route messages directed to that name). Reply-to destinations can always use the local bus name, because the WebSphere MQ link automatically substitutes the virtual queue manager name when passing the message to the WebSphere MQ network.

- While messages are being transported through the WebSphere MQ network, WebSphere MQ treats the names of service integration queue type destinations as WebSphere MQ queue names. This means that WebSphere MQ cannot transport service integration destination names that do not comply with WebSphere MQ queue name restrictions correctly.

  If the target destination name does not comply with WebSphere MQ queue name restrictions, the local bus must define an alias destination that maps the actual bus destination name to a name that does comply with WebSphere MQ queue name restrictions. Alternatively, applications on the local bus can use the WebSphere MQ-compliant name instead of the actual bus destination name.

  In either case, the remote bus must define an alias destination that maps the WebSphere MQ-compliant name to the actual bus destination name. If the reply-to destination name does not comply with WebSphere MQ queue name restrictions, applications on the local bus must use a WebSphere MQ-compliant name instead of the actual bus destination name. The local bus must define an alias destination that maps the WebSphere MQ-compliant name to the actual bus destination name.

While messages are being transported through the WebSphere MQ network, important context information is transported in the MQRFH2 header. You must configure the application so that the MQRFH2 header is included.

Messages with topic style reply-to destinations must have the appropriate publish/subscribe bridge topic mappings defined in the relevant direction so that reply messages can be transferred between a WebSphere MQ network and WebSphere Application Server. This is not automatic, as it is for messages with queue reply destinations.

Messaging between two WebSphere MQ networks through an application server

You can use WebSphere MQ links to send a message from one WebSphere MQ network to another through a WebSphere Application Server application server.

You can exchange messages between two WebSphere MQ networks through an intermediate service integration bus, as shown in the following figure.

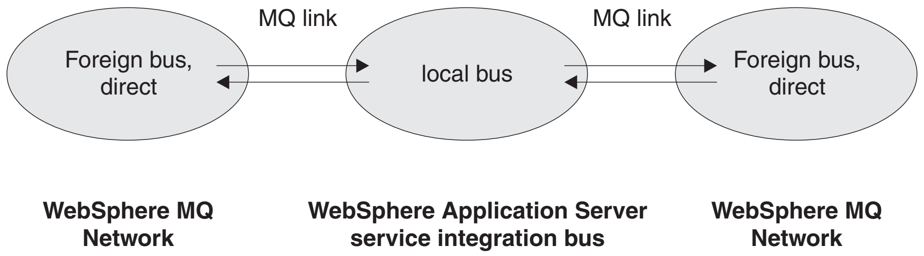


Figure 54. Exchanging messages between two WebSphere MQ networks through an intermediate application server.

Configuration and operation of messaging between two WebSphere MQ networks (buses) through an intermediate service integration bus is much simpler if you choose a service integration bus name that complies with WebSphere MQ queue manager naming restrictions:

- The queue managers in the WebSphere MQ networks “see” the intermediate service integration bus as a queue manager, and refer to it by the virtual queue manager name. Provided that the service integration bus name complies with WebSphere MQ restrictions for queue manager names, it is possible (and highly desirable) for the virtual queue manager name used by both WebSphere MQ networks to be the same as the bus name used by service integration.
- The service integration bus includes two gateway messaging engines, one connecting to each of the WebSphere MQ networks by using WebSphere MQ links (known to the WebSphere MQ gateway queue managers as a WebSphere MQ sender-receiver pair of message channels). The service integration bus

must define the two WebSphere MQ networks as foreign buses with names the same as the WebSphere MQ names for the gateway queue managers.

Messages received by the service integration bus from one WebSphere MQ network and destined for another WebSphere MQ network specify both the target queue name and the target queue manager name. Service integration interprets the target queue manager name as a bus name. When the target queue manager is the gateway queue manager for the target WebSphere MQ network, service integration routes the message correctly. When the target queue manager is not the gateway queue manager for the target WebSphere MQ network, there are two options:

- In service integration, define the target queue manager as an indirect foreign bus, connected by the WebSphere MQ foreign bus defined with the name of the gateway queue manager.
- In service integration, define the target queue (destination name or identifier in service integration terminology) and queue manager (bus name in service integration terminology) combination with an alias destination that maps the combination to the target WebSphere MQ network (foreign bus in service integration terminology) with the destination name (identifier) in the form `target-queue-name@target-queue-manager-name`.

Messages received by the service integration bus from one WebSphere MQ network and destined for another WebSphere MQ network can include a reply-to queue. This is specified as the reply-to queue name and the reply-to queue manager name. When the service integration bus receives the message, the WebSphere MQ link replaces this reply-to information with a service integration bus destination comprising a bus name (which is the WebSphere MQ queue manager name of the gateway queue manager) and a destination name (identifier) of the form `reply-to-queue-name@reply-to-queue-manager-name`. This new reply-to information travels with the message to the receiving WebSphere MQ application. The combined length of the reply-to queue name, the "@" character, and the reply-to queue manager name must be less than or equal to the WebSphere MQ maximum queue name length of 48 characters.

Interoperation using a WebSphere MQ server

A WebSphere MQ server represents a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. Service integration can communicate directly with a WebSphere MQ queue manager or queue sharing group where a WebSphere MQ queue is located. You can configure a service integration queue-type destination to add messages directly onto the WebSphere MQ queue, and retrieve messages directly from the queue.

For interoperation with WebSphere Application Server Version 7.0 or later, the version of WebSphere MQ must be WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ (distributed platforms) Version 7 or later.

To configure WebSphere Application Server for this style of interoperation with WebSphere MQ, you define a WebSphere MQ server. This definition represents the WebSphere MQ queue manager or queue sharing group that hosts the queue which you want to access. The definition has cell scope. The same WebSphere MQ server definition works for all queues in the queue manager or queue sharing group.

Next, you add the WebSphere MQ server as a member of the service integration bus (or buses) that require access to the queues that the WebSphere MQ server hosts. If you have several service integration buses in your cell, you can add the same WebSphere MQ server as a bus member into more than one of these buses.

After you have done this, you can define queue-type destinations in the service integration bus so that service integration adds messages directly onto a WebSphere MQ queue located on the WebSphere MQ server, or retrieves messages directly from that queue, or both. To help you define your service integration destinations, you can (optionally) select the WebSphere MQ queue you want to use from a list which the administrative console gets directly from the WebSphere MQ queue manager or queue sharing group. This facility is called "queue discovery".

If you want to mediate a service integration queue-type destination, then you must define two queues within the destination. One is used to queue messages arriving at the destination ready for mediation; this is called the mediation point. The other is used to queue messages after mediation is complete and the messages are ready to be consumed; this is called the queue point. Either the mediation point, or the queue point, or both, can be defined as WebSphere MQ queues (as previously described).

You can create and configure a WebSphere MQ server by using the administrative console or by using the wsadmin tool. If you use the administrative console, the server creation wizard can automatically discover resources in the WebSphere MQ network.

Network topologies for interoperation using a WebSphere MQ server connection and WebSphere MQ for z/OS shared queues

These examples show simple and complex topologies that enable WebSphere Application Server to interoperate with WebSphere MQ using a WebSphere MQ server connection, and a topology for using WebSphere MQ server connections with WebSphere MQ for z/OS shared queues to create a highly available messaging system.

- “Single WebSphere Application Server application server connected to a single WebSphere MQ queue manager or queue sharing group” on page 434
- “Multiple applications running in separate application servers connected to a WebSphere MQ queue manager” on page 434
- “Using WebSphere MQ for z/OS shared queues with a WebSphere MQ server connection” on page 435

For completeness, the topologies that this topic describes include clustered and highly available topologies. Note that, for clustering and high availability, you need to use the network deployment or z/OS version of the product.

Queue-type destinations with WebSphere MQ server connections

With a regular queue-type destination in the service integration bus, the queue itself is located in a bus member within WebSphere Application Server. The bus member can be an application server or possibly a cluster of application servers. One or more messaging engines in the bus member manage the queue. The messaging engines can put messages onto the queue, get messages from the queue and, if necessary, maintain disk copies of the messages. When an application connects to the service integration bus, it might connect to a messaging engine that is not where the queue is located. In that case, the messaging engine where the application connects communicates with and uses the messaging engine where the queue is located.

With a WebSphere MQ server connection you can configure a service integration bus queue-type destination, so that the queue itself is located in a WebSphere MQ queue manager or queue-sharing group. In this case, the queue manager or queue-sharing group is included in the service integration bus as a bus member. Service integration messaging engines in the bus communicate with and use a WebSphere MQ queue manager to access the queue.

A WebSphere MQ server connection allows applications to perform both get and put operations, unlike a WebSphere MQ link connection which only allows applications to perform put operations.

A WebSphere MQ server connection can use either a “bindings” connection (call attach) or a “client” connection (a TCP/IP connection). A “bindings” connection can only be used when the application server and the queue manager or queue sharing group are running on the same host or in the same logical partition (LPAR). If the application server and the queue manager or queue sharing group are running on different hosts, then a “client” connection must be used.

Single WebSphere Application Server application server connected to a single WebSphere MQ queue manager or queue sharing group

This basic scenario uses a service integration bus with a single messaging engine. The bus includes a queue-type destination configured to use a WebSphere MQ shared queue. A single application connects to the service integration bus and accesses the queue-type destination.

When the application sends a message to the destination, the messaging engine communicates with the WebSphere MQ queue manager and uses it to add the message to the shared queue. When the application receives a message from the destination, the messaging engine communicates with the WebSphere MQ queue manager and uses it to get the message from the shared queue.

When an application is communicating with WebSphere MQ over a WebSphere MQ server connection, it is only aware that it is communicating with a local service integration messaging engine. The messaging engine communicates with WebSphere MQ on behalf of the application. The WebSphere MQ queue manager regards the service integration messaging engine as a WebSphere MQ client.

In the following figure, the connecting line labeled A shows the queue manager appearing to the service integration messaging engine as a member of its local bus. The connecting line labeled B shows the service integration messaging engine appearing to the queue manager as another queue manager.

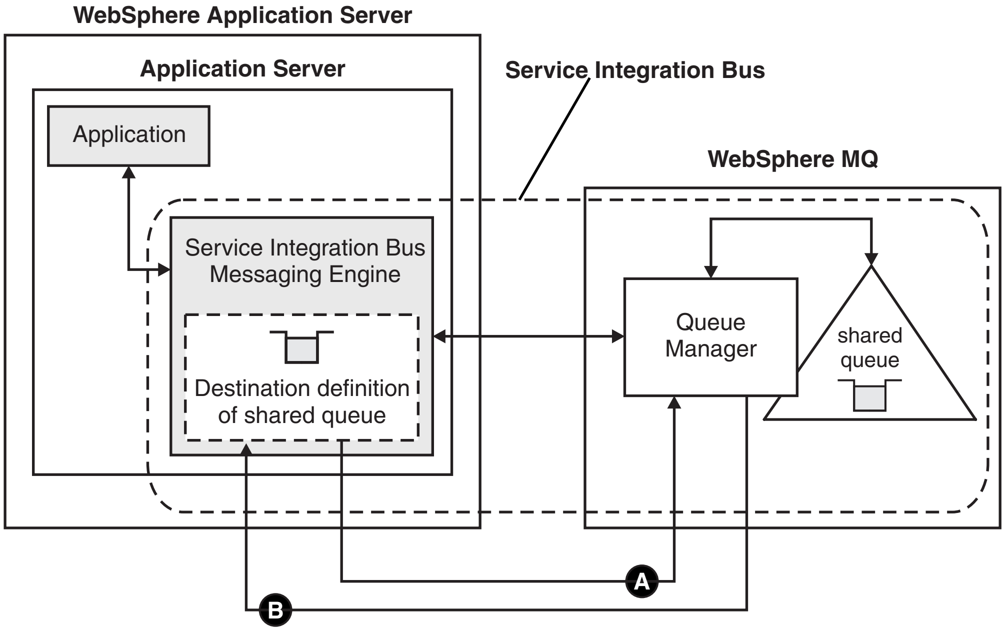

WebSphere Application Server
Application Server
Service Integration Bus
Application
WebSphere MQ
Service Integration Bus Messaging Engine
Queue Manager
shared queue
Destination definition of shared queue
A
B

Figure 55. Single application running within WebSphere Application Server and connecting to WebSphere MQ using a WebSphere MQ server connection.

Multiple applications running in separate application servers connected to a WebSphere MQ queue manager

With a WebSphere MQ server connection, service integration messaging engines dynamically make individual connections to WebSphere MQ queue managers as and when they are required. There are no gateway messaging engines or gateway queue managers as there are when a WebSphere MQ link is used.

The following figure shows two applications running within separate application servers connecting to a WebSphere MQ queue manager over a WebSphere MQ server connection. The service integration bus includes two messaging engines and one queue manager.

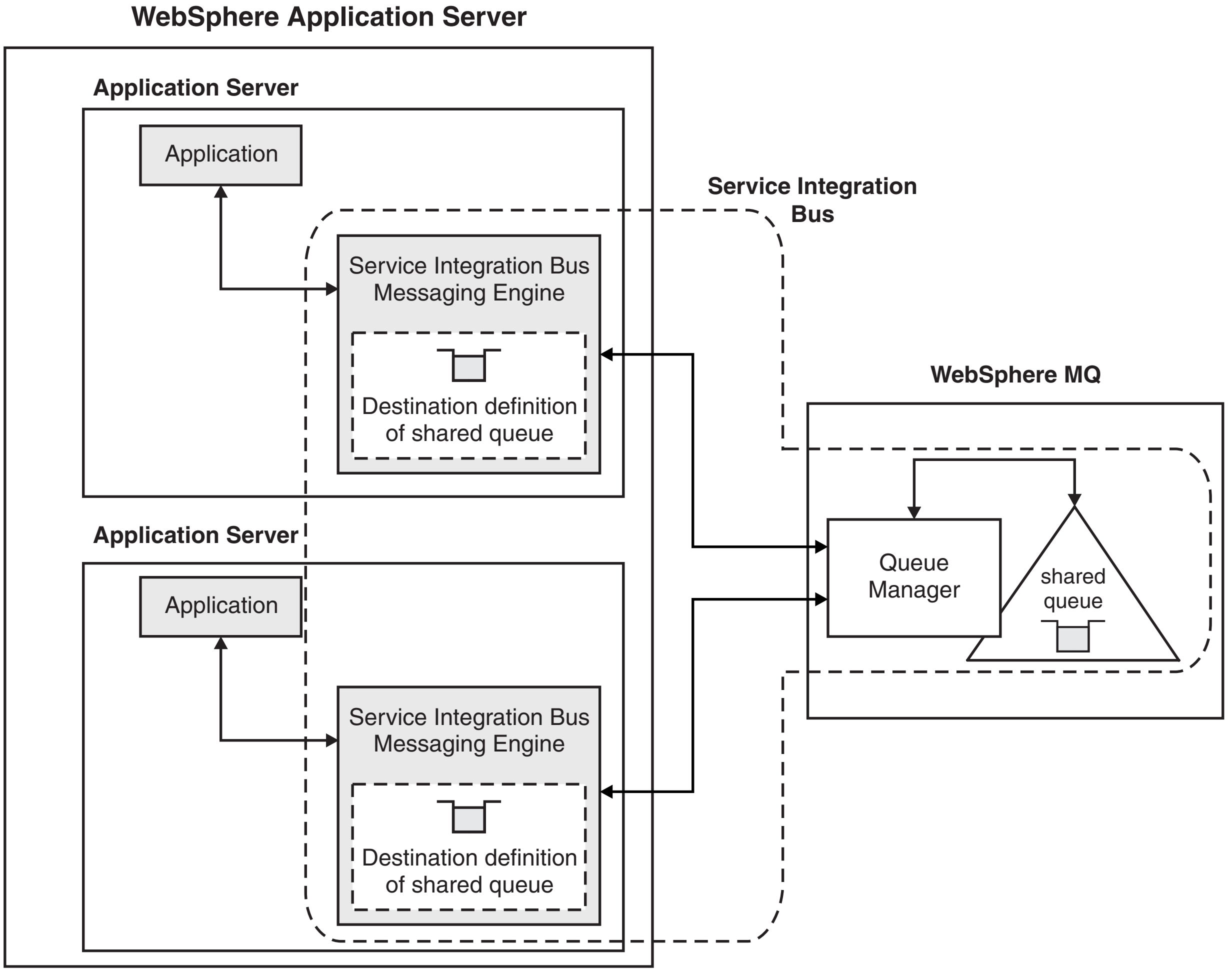

WebSphere Application Server
Application Server
Application
Service Integration Bus Messaging Engine
Destination definition of shared queue
Application Server
Application
Service Integration Bus Messaging Engine
Destination definition of shared queue
Service Integration Bus
WebSphere MQ
Queue Manager
shared queue

Figure 56. Two applications running within separate application servers connecting to a WebSphere MQ queue manager over a WebSphere MQ server connection

Using WebSphere MQ for z/OS shared queues with a WebSphere MQ server connection

WebSphere MQ server connections enable WebSphere Application Server applications to perform get operations (to receive messages from WebSphere MQ queues). You can, therefore, gain benefits by using a WebSphere MQ server to connect to a WebSphere MQ for z/OS queue sharing group. A WebSphere MQ link can connect WebSphere Application Server applications to a queue sharing group, but the applications cannot realize the full benefits of shared queues because they cannot consume messages from them, as aWebSphere MQ link only enables applications to perform put operations.

WebSphere MQ for z/OS queue sharing groups provide significant benefits through the use of shared queues. Multiple applications can send messages to and receive messages from the same shared queue by using different queue managers in the same queue sharing group. This gives the following advantages:

- The different applications (or different instances of the same application) compete to process messages on the same queue. An instance which is able to process messages more quickly, perhaps because the instance is running on a more powerful or less heavily loaded processor, automatically processes a higher proportion of the messages on the queue, giving better utilization of the available resources and better overall response times. This is called “pull workload balancing”.

- If one queue manager in a queue sharing group fails, applications can connect to a different queue manager and continue using the same shared queue. This provides superior availability for your applications. A special feature of queue sharing groups, called “peer level recovery”, handles the cases where an application receives a message from a shared queue but the queue manager fails before processing of the message is complete. Provided that the application is transactional, another queue manager in the same queue sharing group can return the message to the shared queue so that it can be processed without waiting for the failed queue manager to recover. Peer level recovery further enhances the availability of your applications.
- Queue sharing groups also enable service integration to connect to the queue sharing group by using a single network address for the collection of queue managers in the queue sharing group. The connection is automatically redirected to a suitable queue manager in the queue sharing group, based on which queue managers are available, and which is able to provide the best response time. This feature enhances both the availability and the performance of your application.

You can provide these benefits to your service integration applications by defining service integration destinations on shared queues owned by a WebSphere MQ server that is in a queue sharing group. The following figure shows a service integration messaging engine connecting to one queue manager (QM1) in a queue sharing group. The connection enables a service integration application to consume messages from a shared queue. Other service integration applications on the same or a different application server can use different connections (to the same or different queue managers, QM2 or QM3, in the same queue sharing group) to consume messages from the same shared queue.

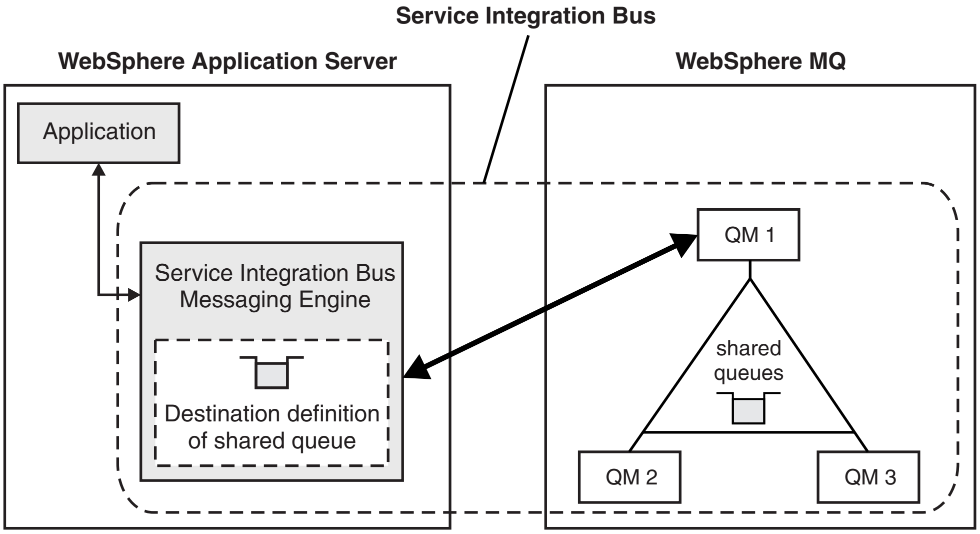

Service Integration Bus
WebSphere Application Server
WebSphere MQ
Application
Service Integration Bus Messaging Engine
Destination definition of shared queue
QM 1
shared queues
QM 2
QM 3

Figure 57. A messaging engine connecting to a queue manager to access a queue sharing group, using a WebSphere MQ server connection

The following figure shows that when a queue manager (QM1) in the queue sharing group is temporarily unavailable, service integration can connect to a different queue manager (QM2), enabling applications to continue processing messages from the queue.

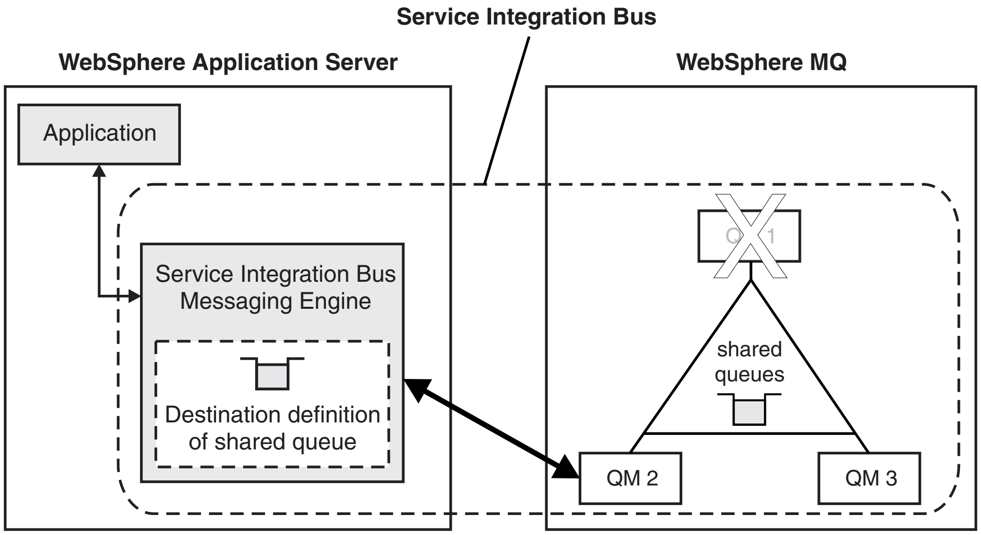

Service Integration Bus
WebSphere Application Server
WebSphere MQ
Application
Service Integration Bus Messaging Engine
Destination definition of shared queue
QM 1
shared queues
QM 2
QM 3

Figure 58. A messaging engine connecting to a second queue manager to access a queue sharing group, after the original queue manger it was using failed

WebSphere MQ queue points and mediation points

A WebSphere MQ queue point is used, by a bus destination on a WebSphere MQ server, to hold messages that are ready to be put onto a WebSphere MQ queue. If messages for the WebSphere MQ queue (or queue-sharing group) are processed by a mediation before being made available to WebSphere MQ, then the service integration destination uses a WebSphere MQ mediation point.

WebSphere MQ queue point

A service integration queue-type destination includes one or more queue points. The destination can use these queue points to hold messages that are ready for a consumer to consume them. When you configure the destination on a bus member that is a WebSphere Application Server single server or cluster, these queue points are service integration message points. When you configure the destination on a bus member that is a WebSphere MQ server, the destination has a single queue point that is a WebSphere MQ queue and is called a WebSphere MQ queue point.

Message consumers, including message-driven beans, receive messages from a queue point. When the queue point is a WebSphere MQ queue point, message consumers receive the messages from the WebSphere MQ queue.

If there is no mediation associated with the destination, and the messages are not redirected to another destination or consumed by a message consumer, then message producers place messages on a queue point. If the queue point is a WebSphere MQ queue point, message producers place messages on the WebSphere MQ queue.

WebSphere MQ mediation point

You can mediate a service integration queue-type destination. When you mediate a destination it is split into two parts called pre-mediated and post-mediated. The pre-mediated part comprises mediation points, the post-mediated part comprises queue points.

The mediation receives messages from the pre-mediated part. If the messages are not redirected to another destination or consumed by a message consumer, the mediation places messages on the post-mediated part. Messages on the post-mediated part are delivered to a message consumer.

When you mediate a destination on a bus member that is a WebSphere MQ server, the destination has a single mediation point that is a WebSphere MQ mediation point, and the post-mediated part is a single queue point that is a WebSphere MQ queue.

WebSphere MQ server and mediated exchange scenarios

When you mediate a service integration bus destination, your mediation runs in a bus member and you specify a combination of mediation points and queue points to handle the messages that are mediated. When you interoperate with WebSphere MQ by using WebSphere MQ server, you can use one of several mediated exchange scenarios.

Queue-type destinations assigned to a WebSphere MQ server bus member can be mediated in the same way as destinations assigned to other bus members. In addition to the mediation task described in Mediating a destination by using a WebSphere MQ queue as the mediation point, WebSphere MQ server supports other mediation scenarios that you also set up by using the administrative console Mediation wizard.

Note: Although WebSphere MQ server extends the way in which queue-type destinations can be mediated, the way in which topic spaces are mediated does not change.

To mediate a service integration bus destination, you must specify a mediation point, a queue point and a mediation execution point:

Mediation point
The location where messages are placed before they are mediated. It can be either a service integration bus member (an application server or a cluster) or a WebSphere MQ queue.

Queue point
The location where messages are placed after they have been mediated. It can be either a service integration bus member (an application server or a cluster) or a WebSphere MQ queue. If there is a default forward routing path and the destination is a queue type destination, the queue point is unused. If the destination is a service type destination, the queue point is absent.

Mediation execution point
The server where the mediation process runs. If the mediation point is a service integration bus member then the mediation execution point is the same bus member as the mediation point.

For more information, see “WebSphere MQ queue points and mediation points” on page 437.

WebSphere MQ server supports the following mediated exchange scenarios:

- Scenario 1: A WebSphere MQ mediation point and a service integration queue point. In this case, you must specify the mediation execution point.
- Scenario 2: A WebSphere MQ mediation point and a WebSphere MQ queue point. In this case, you can use a service integration mediation; you must specify the mediation execution point when you configure the mediation, as for scenario 1.
- Scenario 3: A service integration mediation point and a WebSphere MQ queue point. In this case, you do not have to specify the mediation execution point; WebSphere Application Server automatically allocates the bus member in which the mediation runs.
- Scenario 4: Alternatively, you can use a WebSphere MQ application or a WebSphere Message Broker flow to mediate the destination. In this case, the application or broker flow retrieves messages from the mediation point (which is a WebSphere MQ queue), mediates the messages, then places the mediated messages on the queue point (which is also a WebSphere MQ queue). You do not specify a mediation execution point when you configure the mediation; instead, you specify that there is an external mediation process.

For a queue point, message producers place messages on the queue point and consumers receive messages from the queue point. For a mediation point, message producers place messages on the mediation point. The messages are mediated then put on a post-mediation queue point. Consumers receive messages from the post-mediation queue point.

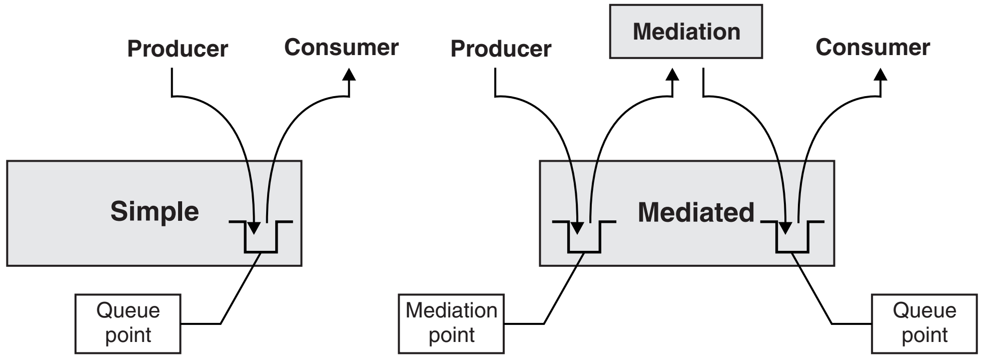


Figure 59. Queue-type destinations assigned to a service integration bus member. Queue points and mediation points are queues of service integration messages held in service integration

For a queue point, message producers place messages on the queue point and consumers receive messages from the queue point. If the queue point is a WebSphere MQ queue point, message producers place messages on the WebSphere MQ queue and consumers receive the messages from the WebSphere MQ queue. For a mediation point, message producers place messages on the mediation point (a WebSphere MQ queue). The messages are mediated, perhaps by an external WebSphere Message Broker flow, then put on a post-mediation queue point (another WebSphere MQ queue). Consumers receive messages from the post-mediation queue point.

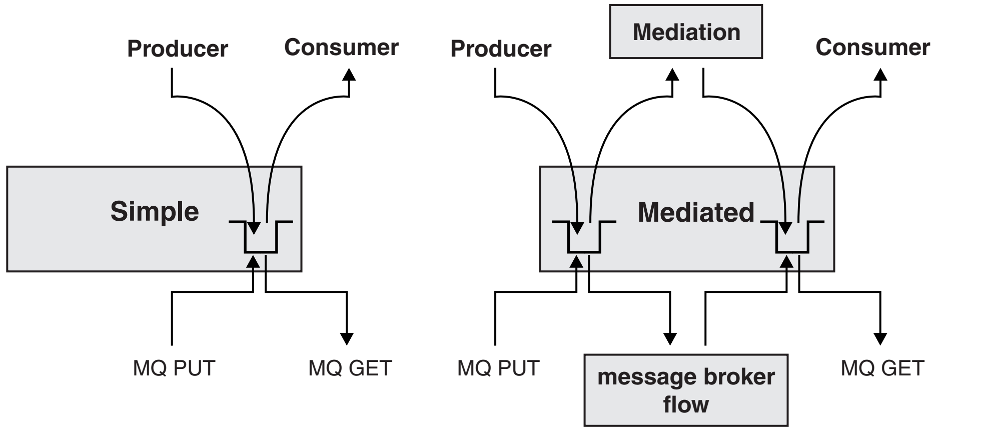


Figure 60. Queue-type destinations assigned to a WebSphere MQ server bus member. Queue points and mediation points can be queues of WebSphere MQ messages held in WebSphere MQ. A WebSphere Message Broker, or other WebSphere MQ application, can run mediations externally.

Scenario 1

In this scenario, you want to mediate a conventional queue-type destination where the queue point is a service integration queue point, and assign a WebSphere MQ queue as the mediation point (the input side of the destination). As the mediation point is a WebSphere MQ queue, a queue point must also be specified.

Messages arriving at the WebSphere MQ queue are processed by the mediation running in an application server. When the messages have been processed by the mediation, they are placed onto the service integration queue point. The mediation itself runs in the service integration bus member that is assigned as the mediation point.

For this scenario, you must complete the following steps using the Mediate destination wizard. These example steps assume that the destination is assigned to a service integration bus member:

1. Navigate to the destinations collection panel for the bus that hosts the destination you want to mediate.
2. Select the queue-type destination that you want to mediate, then click **Mediate**. This starts the Mediate destination wizard.
3. Step 1: Select the mediation that you want to use to mediate the service integration destination.
4. Step 2: Select a WebSphere MQ server bus member to host the mediation point.
5. Step 3: Enter details of the WebSphere MQ queue that will be the mediation point.
6. Step 4: Select a bus member where you want the mediation code to run.
7. Step 5: Review the summary of changes you are about to make, then click **Finish**.

Scenario 2

In this scenario, you want to mediate a WebSphere MQ queue type, with a WebSphere MQ queue point, and assign a WebSphere MQ queue as the mediation point (the input side of the destination). As the mediation point is a WebSphere MQ queue, a mediation execution point must also be specified.

Messages arriving at the destination are processed by the mediation, then placed on the WebSphere MQ queue. The mediation itself runs in the service integration bus member that is assigned as the mediation point.

For this scenario, you must complete the following steps using the Mediate destination wizard. These example steps assume that the destination is assigned to a WebSphere MQ server bus member:

1. Navigate to the destinations collection panel for the bus that hosts the destination you want to mediate.
2. Select the queue-type destination that you want to mediate, then click **Mediate**. This starts the Mediate destination wizard.
3. Step 1: Select the mediation that you want to use to mediate the service integration destination.
4. Step 2: Select a WebSphere MQ server bus member to host the mediation point.
5. Step 3: Enter details of the WebSphere MQ queue that will act as the mediation point.
6. Step 4: Select the service integration bus member where you want the mediation to run.
7. Step 5: Review the summary of changes you are about to make, then click **Finish**.

Scenario 3

In this scenario, you want to mediate a WebSphere MQ queue type and assign a service integration mediation point.

Messages arriving at the destination are processed by the mediation, then placed on the WebSphere MQ queue. The mediation itself runs in the service integration bus member that is assigned as the mediation point.

For this scenario, you must complete the following steps using the Mediate destination wizard. These example steps assume that the destination is assigned to a WebSphere MQ server bus member:

1. Navigate to the destinations collection panel for the bus that hosts the destination you want to mediate.
2. Select the queue-type destination that you want to mediate, then click **Mediate**. This starts the Mediate destination wizard.
3. Step 1: Select the mediation that you want to use to mediate the service integration destination.

4. Step 2: Select the service integration bus member to host the mediation point. The mediation code also runs in this bus member.
5. Step 3: Review the summary of changes you are about to make, then click **Finish**.

Scenario 4

In this scenario, you want to mediate a WebSphere MQ queue type destination and assign a WebSphere MQ queue as the mediation point (the input side of the destination).

The mediation of messages is performed by an external process. Messages arriving at the WebSphere MQ queue are processed by the external process, then placed by the external process on the WebSphere MQ queue-type destination.

For this scenario, you must complete the following steps using the Mediate destination wizard. These example steps assume that the destination is assigned to a WebSphere MQ server bus member:

1. Navigate to the destinations collection panel for the bus that hosts the destination you want to mediate.
2. Select the queue-type destination that you want to mediate, then click **Mediate**. This starts the Mediate destination wizard.
3. Step 1: Select an external process to use for mediating the destination.
4. Step 2: Enter details of the WebSphere MQ queue that you want to act as the mediation point.
5. Step 3: Review the summary of changes you are about to make, then click **Finish**.

WebSphere MQ server: Connection and authentication

Each WebSphere MQ server definition includes the connection properties and authentication settings that service integration uses to connect to the associated WebSphere MQ queue manager or queue-sharing group, either for resource discovery or for messaging.

Connection

Service integration connects to the WebSphere MQ network in the following situations:

- When, as part of the process of creating a WebSphere MQ server by using the administrative console, the automatic resource discovery process runs to capture resource information direct from WebSphere MQ. The wsadmin commands do not support automatic discovery of resources.
- When the WebSphere MQ server is used to pass messages between service integration and WebSphere MQ.

The connection access path is determined by the host, port, transport chain and WebSphere MQ connection channel that you specify when you create the WebSphere MQ server definition. You get this information from the WebSphere MQ system administrator. The connection access path is also affected by the connection mode that you specify:

- You can use client transport mode to establish a TCP/IP network connection between service integration and WebSphere MQ.
- If WebSphere Application Server and WebSphere MQ are co-located on the same system (or, for z/OS systems, on the same partition of the same system) it is more efficient to use bindings transport mode to connect between service integration and WebSphere MQ.

For more information about the mechanisms used to connect to WebSphere MQ for z/OS, see the *z/OS System Setup Guide* in the WebSphere MQ information center.

Authentication

The WebSphere MQ system administrator will probably want service integration to authenticate with WebSphere MQ whenever it connects. This happens whenever message data needs to be exchanged with a queue point or a mediation point that is assigned to a WebSphere MQ server bus member, and when the automated resource discovery process runs while you are configuring a WebSphere MQ server by using the administrative console.

The WebSphere MQ system administrator might also want to set up two different user accounts on the WebSphere MQ system: one with only the privileges needed for resource discovery, and one with only the privileges needed for messaging. The WebSphere MQ server definition supports this requirement by allowing you to configure the MQ server with two authentication aliases, corresponding to these two accounts.

Authentication aliases are restricted to a maximum 12 characters in length, because the user ID that WebSphere MQ uses for checking the identity of new connections also has this restriction. If authentication aliases exceed 12 characters in length, they are truncated.

If you are using Resource Access Control Facility (RACF®) as the security manager on your WebSphere MQ for z/OS system, and using bindings transport mode, you must specify in uppercase characters the user names and passwords for authentication aliases. If you are using RACF and client transport mode, you can specify the user names and passwords in either upper or lowercase characters.

Where an authentication alias exists, the user name and password it contains are examined by WebSphere MQ by using a WebSphere MQ channel security exit. WebSphere MQ for z/OS provides a sample security exit CSQ4BCX3, which demonstrates how you can authenticate based on this information.

When messages are sent to WebSphere MQ for resource discovery, the MQPMO_SET_IDENTITY_CONTEXT option is used. The credentials used to establish a messaging connection must have authority to assert this.

The connection mode you use for connecting to WebSphere MQ affects which credentials are used:

- For a client transport mode connection, the user ID and password from the authentication alias are used by WebSphere MQ. If an authentication alias is not specified in the WebSphere MQ server definition, WebSphere MQ is presented with an empty string for both the user ID and password.
- For a bindings transport mode connection, the credentials associated with the application server processes are used for authentication by WebSphere MQ. Therefore service integration instructs the application server processes to switch credentials and use the user ID and password that exist in the relevant WebSphere MQ server authentication alias. This in turn requires that the application server processes start with sufficient privileges to connect and perform the switch. If an authentication alias is not specified in the WebSphere MQ server definition, a switch of credentials is not attempted and the original credentials of the application server process are used.

Overriding the connection and authentication settings

When you add the WebSphere MQ server definition to a service integration bus to make it a bus member, you can override the server settings and authentication alias used for messaging, with the connection settings and authentication alias used by the bus. You can use this option to create a bus-specific instance of that server and is useful in a multiple bus configuration. Typically you would do this to differentiate connections from different buses and, potentially, to apply different security settings.

User identification

Service integration messages contain two user IDs - a system user ID and an application user ID. WebSphere MQ can set the `user identifier` field of the WebSphere MQ message descriptor (MQMD) from the system user ID used in the service integration message. Additional processing is required to preserve the service integration application user ID when interoperating with WebSphere MQ by using a WebSphere MQ server.

Service integration messages contain two user IDs:

- a system user ID: In general, the system user ID is set to the identity of the user that produced the message, which is specified when the user connects to the bus. The system user ID stored in the message cannot be modified by application code.
- an application user ID: This corresponds to the JMSXUserID message property and can be set by application code.

WebSphere MQ can be configured to set the user identifier field of the WebSphere MQ message descriptor (MQMD) from the system user ID used in the service integration message. However, there is only one field for user IDs in the MQMD. If the destination permits the use of MQRFH2 headers, the application user ID present in the message is placed into the `<sib>` folder of the RFH2 header using a key of `jsApiUserId`.

When a message is received from queue points or mediations points localized on a WebSphere MQ server bus member then, depending on whether the associated WebSphere MQ server definition permits the user IDs to be trusted, the following actions are completed:

- If the WebSphere MQ server is configured to trust user IDs, the system user ID in the service integration message is copied from the user ID in the MQMD.
- If the WebSphere MQ server is not configured to trust user IDs, the system user ID in the service integration message is set to the name of the WebSphere MQ server from which the message has been received.

Consider an example where the following objects have been configured:

- A WebSphere MQ server, QM1
- A WebSphere MQ server bus member with the **`trustUserIds`** attribute set to `FALSE`.
- A queue-type destination, Q1 assigned to the WebSphere MQ server bus member.

If you configured these objects, when a message is received from Q1, the user ID is always set to `QM1` (ignoring the user ID that exists in the message). This happens because the WebSphere MQ server bus member does not trust the user IDs received in inbound messages, instead it always uses the name of the WebSphere MQ server that the message is received from.

Regardless of how the system user ID of the service integration message is set, the application user ID is always set from the jsApiUserId RFH2 value. If this is not present, either because the value pair is not present in the `<sib>` folder of the RFH2 header, or because the message does not have a RFH2 header, then this field will not be set.

As security user IDs are transported in the MQMD message descriptor, they are limited to 12 characters in length. Longer user IDs are truncated.

Request-reply messaging using a WebSphere MQ server

You can provide a reply-to destination in a message sent to a destination that is assigned to a WebSphere MQ server bus member. If the reply comes from a WebSphere MQ application, for example a WebSphere MQ JMS application, some restrictions apply to the reply-to destination. You must also configure a WebSphere MQ link over which the reply can flow.

When a message with a reply-to destination is sent to a destination that is assigned to a WebSphere MQ server bus member, the reply-to destination is represented by the following WebSphere MQ message descriptor fields:

- Queue name: this is set to the name of the service integration destination that has been specified as a reply-to queue.
- Queue manager name: this is set either to the name of the service integration bus from which the message was sent or to the virtual queue manager name specified in the bus member configuration for the MQ server.

Queue names and queue manager names that are not recognized by WebSphere MQ are truncated at the first character that is not a valid WebSphere MQ character, or at the WebSphere MQ limit on the field length.

When you send a message from service integration using a WebSphere MQ server, a WebSphere MQ JMS application can only reply to the reply-to destination in the message when you meet these conditions:

- The reply-to destination name must be a valid WebSphere MQ queue name.
- The reply-to destination must be on a service integration bus that has a name that is a valid WebSphere MQ queue manager name, or the virtual queue manager name specified in the bus member configuration for the MQ server must be a valid WebSphere MQ queue manager name.
- The reply-to destination must be on the same service integration bus as the bus where the message originated.
- You must configure a WebSphere MQ link over which the reply can flow between the service integration bus and the WebSphere MQ network.
- The “Virtual queue manager name” that you allocate to the WebSphere MQ link must match the queue manager name specified for the reply-to destination, which can be either the name of the service integration bus to which the WebSphere MQ link points, or the virtual queue manager name specified in the bus member configuration for the MQ server.

WebSphere MQ server: Transport chain security

System security for a connection between service integration and a WebSphere MQ network is provided by the Transport Level Security (TLS) and Secure Sockets Layer (SSL) protocols.

When WebSphere Application Server uses SSL, the administrator must create an SSL repertoire, a channel and a transport chain. The transport chain must be referenced by the WebSphere MQ server through the server transport chain attribute, and must also be a trusted transport for the service integration bus to which the WebSphere MQ server belongs. The default setting is for service integration buses to trust only the SSL transport.

Two default transport chains are created on each WebSphere MQ server: OutboundBasicWMQClient and OutboundSecureWMQClient. The OutboundSecureWMQClient transport chain uses SSL and is configured to use the server default SSL repertoire. If you want to create your own transport chain, you must define it to every WebSphere MQ server that is a service integration bus member. Here is an example of how you might define your own transport chain by using JACL:

```
wsadmin>tcs = AdminConfig.list("TransportChannelService" ).splitlines()[0]

AdminConfig.create("TCPOutboundChannel" , tcs, [["name" , "MyWMQChain.TCP"]])

wsadmin>ssl=...

wsadmin>AdminConfig.create("SSLOutboundChannel" , tcs , [["name" , "MyWMQChain.SLL"] ,
 ["sslConfigAlias" , "MyRepertoire"]])

wsadmin>rmq=...

wsadmin>AdminConfig.create("RMQOutboundChannel" , tcs , [["name" , "MyWMQChain.RMQ"]])

wsadmin>tcp=...

wsadmin>AdminConfig.create("Chain" , tcs , ["name" , "MyWMQChain"] , ["enable" , "true"] ,
 ["transportChannels", [rmg , ssl , tcp]])
```

This example creates a transport chain suitable for connecting a WebSphere MQ server to WebSphere MQ by using SSL. The chain is called MyWMQChain, and uses an SSL repertoire called MyRepertoire.

WebSphere MQ uses a single cipher suite only for securing connections to a queue manager, although WebSphere Application Server SSL repertoires allow you to specify multiple cipher suites. Each cipher suite is tried sequentially until a successful connection is established, or until all the cipher suites have been tried. The most recent cipher suite that allowed a successful connection is cached on a WebSphere MQ server bus member basis, and is tried first on subsequent connection attempts.

When transport security is enabled, the transport chain used for connections to WebSphere MQ must be a permitted chain otherwise it is not possible to establish a connection to WebSphere MQ.

WebSphere MQ server: Restrictions with mixed level cells and clusters

If you are using a WebSphere MQ server with cells or clusters that include more than one version of WebSphere Application Server, you might need to be aware of the application server versions when you deploy applications that communicate with WebSphere MQ queues.

With a WebSphere MQ server, applications reference service integration destinations that have queue points or mediation points set as WebSphere MQ queues. These service integration destinations are available to Version 6.1 and later versions of WebSphere Application Server, but are not available to Version 6.0 and earlier versions of WebSphere Application Server, which do not support them. There are some restrictions on the deployment of an application that references a service integration destination that is not available to earlier versions of WebSphere Application Server.

Take the following example situation:

- You define a service integration destination destination_x that has a queue point set as a WebSphere MQ queue.
- Destination_x is in bus bus_y.
- Destination_x is visible in all application servers at Version 6.1 or later, but it is not visible in application servers at Version 6.0 or earlier.
- You want to deploy an application appl_z that references destination_x.

In this example situation, the following restrictions apply:

- You must not deploy appl_z on any application server that is a member of bus_y unless that application server is at Version 6.1 or later.
- You must not deploy appl_z on any WebSphere Application Server cluster that is a member of bus_y unless every application server in that cluster is at Version 6.1 or later.
- You can deploy appl_z on any application server or cluster that is not a member of bus_y. However, if any of the application servers in that cell are at Version 6.0 or earlier, the following additional rules apply:
  - You must define a target group containing messaging engines in bus_y that always run in application servers at Version 6.1 or later.
  - You must configure the JMS connection factory (if any) used by appl_z to select that target group with Target significance `Required`. You must configure the JMS activation specification (if any) for appl_z to select that target group with Target significance `Required`

Message-driven beans - automatic message retrieval

WebSphere Application Server supports the use of message-driven beans as asynchronous message consumers.

The following figure shows an incoming message being passed automatically to the onMessage() method of a message-driven bean that is deployed as a listener for the destination. The message-driven bean processes the message, in this case passing the message on to a business logic bean for business processing.

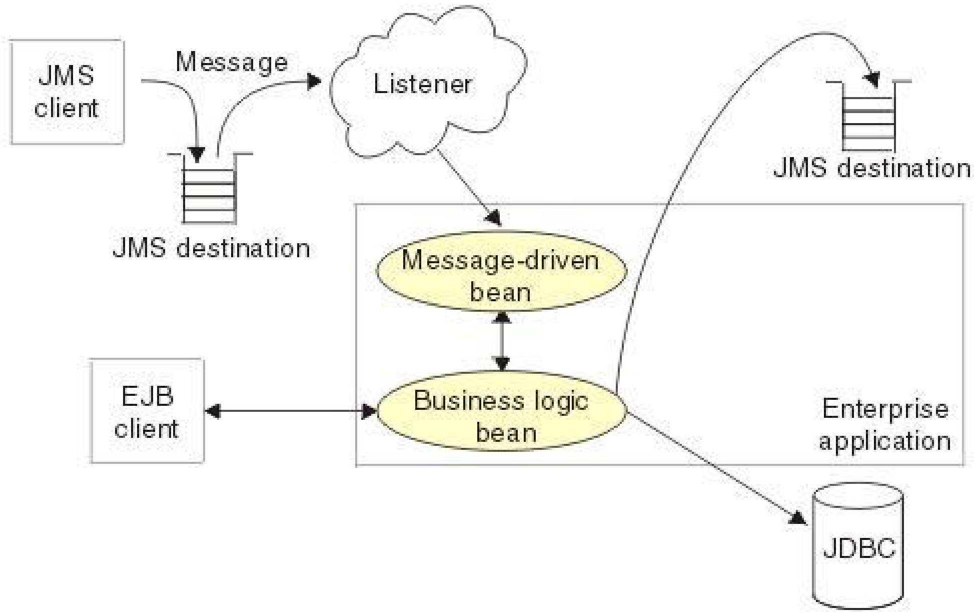


Figure 61. Messaging with message-driven beans

A client sends messages to the destination (or endpoint) for which the message-driven bean is deployed as the message listener. When a message arrives at the destination, the EJB container invokes the message-driven bean automatically without an application having to explicitly poll the destination. The message-driven bean implements some business logic to process incoming messages on the destination.

It can be helpful to separate the business logic of your application from the communication interfaces, such as the JMS request and response handling. To achieve this separation, you can design your message-driven bean to delegate the business processing of incoming messages to another enterprise bean. Separating message handling and business processing enables different users to access the same business logic in different ways, either through incoming messages or, for example, from a WebSphere J2EE client.

Message-driven beans can be configured as listeners on a Java EE Connector Architecture (JCA) 1.5 or 1.6 resource adapter or against a listener port. With a JCA 1.5 resource adapter, message-driven beans can handle generic message types, not just JMS messages. This makes message-driven beans suitable for handling generic requests inbound to WebSphere Application Server from enterprise information systems through the resource adapter. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or just *endpoints*.

All message-driven beans must implement the MessageDrivenBean interface. For JMS messaging, a message-driven bean must also implement the message listener interface, javax.jms.MessageListener.

Messages arriving at a destination being processed by a message-driven bean have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see Securing enterprise bean applications.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, for example the default messaging provider that is part of WebSphere Application Server or the WebSphere MQ messaging provider. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA 1.5-compliant resources, to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, for example the V5 default messaging provider, you must configure JMS message-driven beans against a listener port.

Message-driven beans, activation specifications, and listener ports

Guidelines, related to versions of WebSphere Application Server, to help you choose when to configure your message-driven beans to work with listener ports rather than activation specifications.

You can configure the following resources for message-driven beans:

- Activation specifications for message-driven beans that comply with Java EE Connector Architecture (JCA) Version 1.5.
- The message listener service, listener ports, and listeners for any message-driven beans that you want to deploy against listener ports.

Activation specifications are the standardized way to manage and configure the relationship between an MDB running in WebSphere Application Server and a destination in WebSphere MQ. They combine the configuration of connectivity, the Java Message Service (JMS) destination and the runtime characteristics of the MDB, within a single object.

Activation specifications supersede the use of listener ports, which became a stabilized feature in WebSphere Application Server Version 7.0 (for more information, see “Stabilized features” on page 1137). There are several advantages to using activation specifications over listener ports:

- Activation specifications are simple to configure, because they only require two objects: the activation specification and a message destination. Listener ports require three objects: a connection factory, a message destination, and the message listener port itself.
- Activation specifications are not limited to the server scope. They can be defined at any administrative scope in WebSphere Application Server. Message listener ports must be configured at the server scope. This means that each server in a node requires its own listener port. For example, if a node is made up of three servers, three separate listener ports must be configured. Activation specifications can be configured at the node scope, so in the example only one activation specification would be needed.
- Activation specifications are part of the Java Platform, Enterprise Edition Connector Architecture 1.5 and 1.6 standards specification (JCA 1.5 and 1.6). Listener port support in WebSphere Application Server makes use of the application server facilities interfaces defined in the JMS specification, but is not part of any specification itself.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

If you want to use message-driven beans with a messaging provider that does not have a JCA Version 1.5 or 1.6 resource adapter, you cannot use activation specifications and therefore you must configure your beans against a listener port. There are also a few scenarios in which, although you could use activation specifications, you might still choose to use listener ports. For example, for compatability with existing message-driven bean applications. Here are some guidelines, related to versions of WebSphere Application Server, to help you choose when to use listener ports rather than activation specifications:

- WebSphere Application Server Version 4 does not support message-driven beans, so listener ports and activation specifications are not applicable. WebSphere Application Server Version 4 does support message beans, but these are not message-driven beans.
- WebSphere Application Server Version 5 supports EJB 2.0 (JMS only) message-driven beans that are deployed using listener ports. This deployment technology is sometimes called application server facility (ASF).

- WebSphere Application Server Version 6 continues to support message-driven beans that are deployed to use listener ports, and also supports JCA, which you can use to deploy message-driven beans that use activation specifications. This gives you the following options for deploying message-driven beans on WebSphere Application Server Version 6:
  - You must deploy default messaging (service integration bus) message-driven beans to use activation specifications.
  - You must deploy WebSphere MQ message-driven beans to use listener ports.
  - You can deploy third-party messaging message-driven beans to use either listener ports or activation specifications, depending on the facilities available from your third-party messaging provider.
- WebSphere Application Server Version 7.0 or later continues to support the same options for message-driven bean deployment that WebSphere Application Server Version 6 supports, and adds a new option for WebSphere MQ message-driven beans. This gives you the following options for deploying message-driven beans on Version 7.0 or later:
  - You must deploy default messaging (service integration bus) message-driven beans to use activation specifications.
  - You can deploy new and existing WebSphere MQ message-driven beans to use listener ports (as on WebSphere Application Server Version 6) or to use activation specifications.
  - You can deploy third-party messaging message-driven beans to use either listener ports or activation specifications, depending on the facilities available from your third-party messaging provider.

To assist in migrating listener ports to activation specifications, the WebSphere Application Server administrative console provides a **Convert listener port to activation specification** wizard on the Message listener port collection panel. This allows you to convert existing listener ports into activation specifications. However, this function only creates a new activation specification with the same configuration used by the listener port. It does not modify application deployments to use the newly created activation specification.

Message processing in ASF mode and non-ASF mode

Application Server Facilities (ASF) mode is the default method by which the message listener service in WebSphere Application Server processes messages. This topic explains how WebSphere Application Server processes messages in ASF mode and how it processes messages when ASF mode is turned off.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

Main features of ASF mode

By default, message-driven beans (MDBs) that are deployed on WebSphere Application Server for use with listener ports, use ASF mode to monitor JMS destinations and to process messages.

IBM i In ASF mode, a thread is allocated for work when a message is detected at the destination for it to process. The number of threads that can be active concurrently is dictated by the value specified for the `Maximum Sessions` property for the listener port.

IBM i In client connection (socket attach) mode, each active thread is an individual physical network connection. You should keep this in mind when you are deciding whether to use ASF or non-ASF mode in your configuration. If you are using WebSphere MQ Version 7.x as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

If WebSphere MQ is your messaging provider, there are several configurations you can use in ASF mode. With the following configurations each thread uses a separate physical network connection:

- A WebSphere MQ Version 6.0 queue manager.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `6`.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `0`.

With the following configuration, threads share a user-defined number of physical network connections:

- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `1` or higher. In this case each thread represents an individual connection to a queue manager. However, each thread does not have its own physical network connection, Instead, the threads share the number of network connections specified in the **SHARECNV** (sharing conversations) parameter.

IBM i

Main features of non-ASF mode

In non-ASF mode threads are active from the moment that the listener port is turned on. The number of active threads is dictated by the value specified for the **Maximum Sessions** property on the listener port. The number of threads specified in **Maximum Sessions** are active, regardless of the number of messages that are available to be processed.

In non-ASF mode, when a listener port browses for messages at the destination, it will take the message that is first in the queue at the destination for processing. This means that messages are processed close to the order in which they arrive at the destination.

In client connection (socket attach) mode, each active thread is an individual physical network connection. You should keep this in mind when you are deciding whether to use ASF or non-ASF mode in your configuration. If you are using WebSphere MQ Version 7.x as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

If WebSphere MQ is your messaging provider, there are several configurations you can use in non-ASF mode. With the following configurations each thread uses a separate physical network connection:

- A WebSphere MQ Version 6.0 queue manager.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `6`.
- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `0`.

With the following configuration, threads share a user-defined number of physical network connections:

- A WebSphere MQ Version 7.x queue manager, using a connection factory that has the **Provider version** property set to `7` or `unspecified`, connecting over a WebSphere MQ channel that has the **SHARECNV** (sharing conversations) parameter set to `1` or higher. In this case each thread represents an individual connection to a queue manager. However, each thread does not have its own physical network connection. Instead, the threads share the number of network connections specified in the **SHARECNV** (sharing conversations) parameter.

Note: Non-ASF mode cannot be selected on z/OS systems.

How messages are processed in ASF mode

In ASF mode, server sessions and threads are only allocated for work when a message that is suitable for the message-driven bean (MDB) is detected. The number of threads that an MDB can process concurrently is determined by the value of the **Maximum Sessions** property for the listener port.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

IBM i

The following diagram shows how messaging takes place between WebSphere Application Server and WebSphere MQ when the message listener service is operating in ASF mode.

Figure 62. Message processing in ASF mode on distributed and IBM i systems

As shown in the diagram, when the message listener service is operating in ASF mode, messages are processed in the following way:

1. When the listener port is started, it opens a connection to the WebSphere MQ queue manager and creates an internal queue agent.
2. The queue agent listens to the JMS destination for messages.
3. The queue agent detects a message.
4. The queue agent checks whether the message is suitable for the MDB that is using the listener port.
5. If the message is suitable for the MDB, the listener port allocates a thread from the message listener service thread pool, and allocates a server session from the application server's server session pool. If this is the first time the server session has been used since the listener port has been started, it opens a connection to the JMS provider. The allocated server session runs on the allocated thread.
6. The queue agent passes the ID of the message to the server session. It then starts listening for messages again.
7. The server session uses the message ID to retrieve the message from the destination.
8. The server session processes the message by calling the `onMessage()` method of the MDB.
9. When the message has been processed, the server session exits and returns to the application server session pool. The connection that the server session opened to the JMS provider remains open so that the server session does not need to re-establish the connection the next time it is used.
10. The thread exits and returns to the message listener service thread pool.

The number of threads that an MDB can process concurrently is determined by the value of the **Maximum Sessions** property for the listener port. If you set **Maximum Sessions** to the default value of `1`, this means that the MDB can only process one message at a time. Therefore, if the queue agent finds a second message before the first message has finished being processed, the queue agent blocks the second message until processing of the first message is complete and the server session has become available.

If you want to process more than one message concurrently, you can do this in ASF mode by setting **Maximum Sessions** to a value higher than `1`. For example, if you set **Maximum Sessions** to `2`, messages are processed in the following way:

1. The queue agent detects the first message and allocates a thread and a server session as in the first example. The message is processed using the `onMessage()` method of the MDB.

2. Whilst the first message is processing, the queue agent starts listening for messages again.
3. The queue agent detects the second message and allocates a second thread and a second server session. The message is processed using the `onMessage()` method of the MDB.
4. When the first message is processed, the first server session exits and returns to the server session pool. The first thread exits and returns to the thread pool.
5. When the second message is processed, the second server session exits and returns to the server session pool. The second thread exits and returns to the thread pool.

How messages are processed in non-ASF mode

IBM i

In non-ASF mode threads are active from the moment that the listener port is started. The number of active threads is dictated by the value specified for **`Maximum Sessions`**. The number of threads specified in **`Maximum Sessions`** are active, regardless of the number of messages that are available to be processed. Each active thread is an individual physical network connection.

If you are using WebSphere MQ Version 7.0 or later as your messaging provider, it is possible to have up to ten threads sharing a single physical network connection.

For WebSphere Application Server Version 7 and later, listener ports are stabilized. For more information, read the article on stabilized features. You should plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, you should not begin this migration until you are sure the application does not have to work on application servers earlier than WebSphere Application Server Version 7. For example, if you have an application server cluster with some members at Version 6.1 and some at a later version, you should not migrate applications on that cluster to use activation specifications until after you migrate all the application servers in the cluster to the later version.

Message processing in non-ASF mode

You activate non-ASF mode by specifying a non-zero value for the **`NON.ASF.RECEIVE.TIMEOUT`** message listener service custom property. **`NON.ASF.RECEIVE.TIMEOUT`** acts as a switch that turns off ASF mode, and also as a timeout value for the `receive()` method.

Note: The following message listener service custom properties do not work in non-ASF mode:
- **`SERVER.SESSION.POOL.REAP`**
- **`SERVER.SESSION.POOL.UNUSED.TIMEOUT`**
- **`SERVER.SESSION.POOL.UNUSED.TIMEOUT.lpaname`**

The following diagram shows how message processing takes place between WebSphere Application Server and WebSphere MQ in non-ASF mode:

IBM i

Figure 63. Message processing in non-ASF mode

As shown in the diagram, when the message listener service is operating in non-ASF mode, messages are processed in the following way:
1. When the listener port is started, it gets one thread from the message listener service thread pool.
2. The listener port opens a connection to the WebSphere MQ queue manager on the thread and creates a JMS message consumer. The message consumer listens to the JMS destination which the listener port is configured to listen to.
3. The listener port creates a transaction to manage the message processing.

4. The thread calls the `receive()` method on the message consumer to listen for messages at the destination. If the `receive()` method does not detect a message in the time specified for **NON.ASF.RECEIVE.TIMEOUT**, the application server rolls back the active transaction and starts a new one. The thread then starts calling the `receive()` method again.
5. When the message consumer detects a message it checks whether the message is suitable for the MDB that is using the listener port.
6. If the message is suitable, the `receive()` method takes it off the destination and sends it to the thread.
7. The thread invokes the `onMessage()` method of the MDB on the message consumer, and the message is processed.
8. If the message finishes processing successfully, the transaction commits. If the message does not process successfully, the transaction rolls back.
9. A new transaction is started and the message consumer calls the `receive()` method to listen for new messages.

The number of threads that an MDB can process concurrently is determined by the value of the **Maximum Sessions** property for the listener port. If you set **Maximum Sessions** to the default value of `1`, this means that the MDB can only process one message at a time. If you want to process more than one message concurrently, you can do this in ASF mode by setting **Maximum Sessions** to a value higher than `1`. For example, if you set **Maximum Sessions** to `2`, messages are processed in the following way:

1. When the listener port is started, it gets two threads from the message listener service thread pool.
2. The listener port creates a message consumer and a transaction on each thread. The message consumers listen to the destination which the listener port is configured to listen to.
3. Both message consumers call the `receive()` method to listen for messages on the destination. The consumers compete to get messages from the destination.
4. When one of the consumers successfully retrieves the message, it processes it by calling the `onMessage()` method of the MDB. The other message consumer keeps on calling the `receive()` method to listen for messages on the destination.

How to avoid unwanted transaction timeouts

If your messaging system is running in non-ASF mode, to avoid unwanted transaction timeouts, you must allow a sufficient amount of time for processing to be completed before the total transaction lifetime timeout is reached. Therefore, you must make sure that the value that you specify for the **NON.ASF.RECEIVE.TIMEOUT** message listener service custom property is smaller than the value that you specify for the **Total transaction lifetime timeout** transaction service property, and also that the difference between the values of the two properties is greater than the amount of time that the `onMessage()` method of the message-driven bean (MDB) takes to process the message.

As the following example shows, if these properties are not correctly configured, transactions can time out before they are completed. This is because the thread begins calling the `receive()` method as soon as the transaction is created. In the following example, **NON.ASF.RECEIVE.TIMEOUT** is set to `110000` milliseconds (110 seconds), **Total transaction lifetime timeout** is set to `120` seconds and the onMessage () method of the MDB takes 15 seconds to process a message. The example supposes that a message does not appear at the destination until the `receive()` method has almost timed out:

1. The listener port starts. It allocates a thread from the thread pool and creates a transaction and a message consumer on the thread.
2. The thread calls the `receive()` method to listen for messages.
3. After 110 seconds a message appears at the destination.
4. The thread removes the message from the destination and calls the `onMessage()` method of the MDB to begin processing the message.
5. 10 seconds later, the transaction timeout is reached. The application server marks the transaction for rollback.

6. 5 seconds later, the `onMessage()` method finishes processing the message and tries to commit the transaction.
7. The total amount of time that has elapsed since the transaction was started is 125 seconds (110 seconds waiting for a message, plus 15 seconds to process the message). As this is longer than the transaction timeout, the application server prevents the transaction from being committed, and it is rolled back.

For further information about how to configure the **`NON.ASF.RECEIVE.TIMEOUT`** and **`Total transaction lifetime timeout`** properties to avoid unwanted transaction time outs, see the related tasks.

Message-driven beans - JCA components

There are several administrative components that you configure for message-driven beans as listeners on a Java EE Connector Architecture (JCA) 1.5 resource adapter.

Components for a JCA resource adapter

When a resource adapter is installed, it provides definitions and classes for administered objects such as activation specifications. The administrator creates and configures activation specifications with Java Naming and Directory Interface (JNDI) names that are then available for applications to use.

The JCA resource adapter uses an activation specification to configure a particular endpoint. Each application that configures one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application uses the activation specification to provide configuration properties for the processing of inbound messages.

JMS components used with a JCA messaging provider

Message-driven beans that implement the javax.jms.MessageListener interface can be used with JMS messaging.

An application that uses JMS messaging needs access at runtime to configured objects such as connection factories and destinations:

- When the JMS provider is the default JMS provider or the WebSphere MQ messaging provider, the administrator configures these objects for the JMS provider. For example, to configure a JMS activation specification for the WebSphere MQ messaging provider, in the WebSphere Application Server administrative console navigate to **Resources** > **JMS->Activation specifications**.
- Otherwise the administrator configures these objects for the JMS resource adapter, which connects the application to a JMS provider, by navigating to **Resources** > **Resource Adapters**.

If the application contains one or more message-driven beans, the administrator must configure either a JMS activation specification or a message listener port. For JCA-compliant messaging providers, the administrator usually configures an activation specification. But for the WebSphere MQ messaging provider there is a choice; the administrator can configure an activation specification or, for compatibility with previous versions of WebSphere Application Server, the administrator can configure a message listener port.

The JMS activation specification provides the deployer with information about the configuration properties of a message-driven bean related to the processing of the inbound messages. For example, a JMS activation specification specifies the name of the service integration bus to connect to, information about the message acknowledgement modes, message selectors, destination types, and whether durable subscriptions are shared across connections with members of a server cluster.

The activation specification identifies a JMS destination by specifying its JNDI name. The message-driven bean acts as a listener on a specific JMS destination.

The JMS destination refers to a service integration bus destination (or WebSphere MQ destination) that the administrator must also configure. For more information about JMS resources and service integration, see “Default messaging” on page 350.

J2C activation specification configuration and use

Configure J2C activation specifications, and use them in the deployment of message-driven beans for JCA 1.5 resources.

J2C activation specifications are part of the configuration of inbound messaging support that can be part of a JCA 1.5 resource adapter. Each JCA 1.5 resource adapter that supports inbound messaging defines one or more types of message listener in its deployment descriptor (`messagelistener` in the ra.xml). The message listener is the interface that the resource adapter uses to communicate inbound messages to the message endpoint. A message-driven bean (MDB) is a message endpoint and implements one of the message listener interfaces provided by the resource adapter. By allowing multiple types of message listener, a resource adapter can support a variety of different protocols. For example, the interface javax.jms.MessageListener, is a type of message listener that supports JMS messaging. For each type of message listener that a resource adapter implements, the resource adapter defines an associated activation specification (`activationspec` in the ra.xml). The activation specification is used to set configuration properties for a particular use of the inbound support for the receiving endpoint.

When an application containing a message-driven bean is deployed, the deployer must select a resource adapter that supports the same type of message listener that the message-driven bean implements. As part of the message-driven bean deployment, the deployer needs to specify the properties to set on the J2C activation specification. Later, during application startup, a J2C activation specification instance is created, and these properties are set and used to activate the endpoint (that is, to configure the resource adapter inbound support for the specific message-driven bean).

Applications with message-driven beans can also specify all, some, or none of the configuration properties needed by the ActivationSpec class, to override those defined by the resource adapter-scoped definition. These properties, specified as activation-config properties in the deployment descriptor for the application, are configured when the application is assembled. To change any of these properties requires redeploying the application. These properties are unique to this applications use and are not shared with other message-driven beans. Any properties defined in the application deployment descriptor take precedence over those defined by the resource adapter-scoped definition. This allows application developers to choose the best defaults for their applications.

Message-driven beans - transaction support

Message-driven beans can handle messages on destinations (or endpoints) within the scope of a transaction.

Transaction handling when using the Message Listener Service with WebSphere MQ JMS

There are three possible cases, based on the message-driven bean deployment descriptor setting you choose: container-managed transaction (required), container-managed transaction (not supported), and bean-managed transaction.

In the message-driven bean deployment descriptor settings, you can choose whether the message-driven bean manages its own transactions (bean-managed transaction), or whether a container manages transactions on behalf of the message-driven bean (container-managed transaction). If you choose container-managed transactions, in the deployment descriptor notebook, you can select a container transaction type for each method of the bean to determine whether container transactions are required or not supported. The default container transaction type is required.

Container-managed transaction (required)

In this case, the application server starts a global transaction before it reads any incoming message from the destination, and before the `onMessage()` method of the message-driven bean is invoked by the application server. This means that other EJBs that are invoked in turn by the message, and interactions with resources such as databases can all be scoped inside this single global transaction, within which the incoming message was obtained.

If this application flow completes successfully, the global transaction is committed. If the flow does not complete successfully, (if the transaction is marked for rollback or if a runtime exception occurs), the transaction is rolled back, and the incoming message is rolled back onto the message-driven bean destination.

Container-managed transaction (not supported)

In this case there is no global transaction, but the JMS provider can still deliver a message from a message-driven bean destination to the application server in a unit of work. You can consider this as a local transaction, because it does not involve other resources in its transactional scope.

The application server acknowledges message delivery on successful completion of the `onMessage()` dispatch of the message-driven bean (using the acknowledgement mode specified by the assembler of the message-driven bean).

However, the application server does not perform an acknowledge, if an unchecked runtime exception is thrown from the `onMessage()` method. So, does the message roll back onto the message-driven bean destination (or is it acknowledged and deleted)?

The answer depends on whether a syncpoint is used by your JMS provider and can vary depending on the operating platform (in particular the z/OS operating platform can impart different behavior here).

If your JMS provider establishes a syncpoint around the message-driven bean message consumption in this container-managed transaction (not supported) case, the message is rolled back onto the destination after an unchecked exception.

If a syncpoint is not used, then the message is deleted from the destination after an unchecked exception.

For related information, see the technote “MDB behavior is different on z/OS than on distributed when getting nonpersistent messages within syncpoint” at http://www.ibm.com/support/docview.wss?uid=swg21231549.

Bean-managed transaction

In this case, the action is similar to the container-managed transaction (not supported) case. Even though there might be a user transaction in this case, any user transaction started within the onMessage dispatch of the message-driven bean does not include consumption of the message from the message-driven bean destination within the transaction scope. To do this, use the container-managed transaction (required) scenario.

Message redelivery

In each of the previous three cases, a message that is rolled back onto the message-driven bean destination is eventually re-dispatched. If the original rollback was due to a temporary system problem, you would expect the re-dispatch of the message-driven bean with this message to succeed on re-dispatch. If, however, the rollback was due to a specific message-related problem, the message would repeatedly be rolled back and re-dispatched. This is known as a poison message scenario.

If your messaging system uses listener ports, the application server handles this scenario, by tracking the frequency with which a specific message is dispatched, and by stopping the associated listener port after a specified number of attempted re-deliveries of that message have occurred.

If your messaging system uses listener ports, you can avoid a poison message scenario by configuring the following property:

Maximum Retries
The **Maximum Retries** parameter specifies the number of times that the listener tries to deliver a specific message to a message-driven bean instance before the listener is stopped.

If this parameter is set to `0` the listener port will stop after a single failure of a message to be delivered successfully.

For more information about this property, see Listener port settings.

If your messaging system uses activation specifications, the poison message scenario is handled in a slightly different way. Whereas listener ports track the number of times a specific message has failed and been re-delivered, activation specifications count the number of sequential message delivery failures.

If your messaging system uses the default messaging provider (service integration), you must configure the following properties on your activation specification to avoid a poison message scenario:

Automatically stop endpoints on repeated message failure
You must make sure that this option is selected.

This property suspends message delivery to the endpoint when the **Sequential failed message threshold** is reached.

Sequential failed message threshold
This parameter determines how many message deliveries can fail before message delivery is suspended.

To enable this parameter you must have the **Automatically stop endpoints on repeated message failure** option selected.

Delay between failing message retries
This parameter specifies how much time must elapse before a message which has failed to be delivered successfully, is re-delivered.

If you specify `0` for this parameter there will be no delay before a message is re-delivered.

To enable this parameter you must have the **Automatically stop endpoints on repeated message failure** option selected.

For more information on these properties, see JMS activation specification [Settings].

If your messaging system uses the WebSphere MQ messaging provider, you must configure the following properties on your activation specification to avoid a poison message scenario:

Stop endpoint if message delivery fails
You must make sure that this option is selected.

This property suspends message delivery to the endpoint when the **Number of sequential delivery failures before suspending endpoint** is reached.

Number of sequential delivery failures before suspending endpoint
This parameter determines how many message deliveries can fail before message delivery is suspended.

To enable this parameter you must have the **Stop endpoint if message delivery fails** option selected.

For more information on these properties, see WebSphere MQ messaging provider activation specification advanced properties

As an alternative to relying on your application server to stop the listener port or activation specification if a poison message scenario occurs, you can configure WebSphere MQ to resolve the problem. In WebSphere MQ specify a backout queue (**BOQUEUE**), and a backout threshold value (**BOTHRESH**). If you do this, WebSphere MQ handles the poison message. For more information about handling poison messages, see the WebSphere MQ *Using Java* section of the WebSphere MQ library.

Message-driven beans - listener port components

The WebSphere Application Server support for message-driven beans deployed against listener ports is based on JMS message listeners and the message listener service, and builds on the application server facility (ASF) support in the JMS provider.

Note: From WebSphere Application Server Version 7, listener ports are stabilized. For more information, read the article on stabilized features. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the WebSphere MQ messaging provider, refer to related tasks.

The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

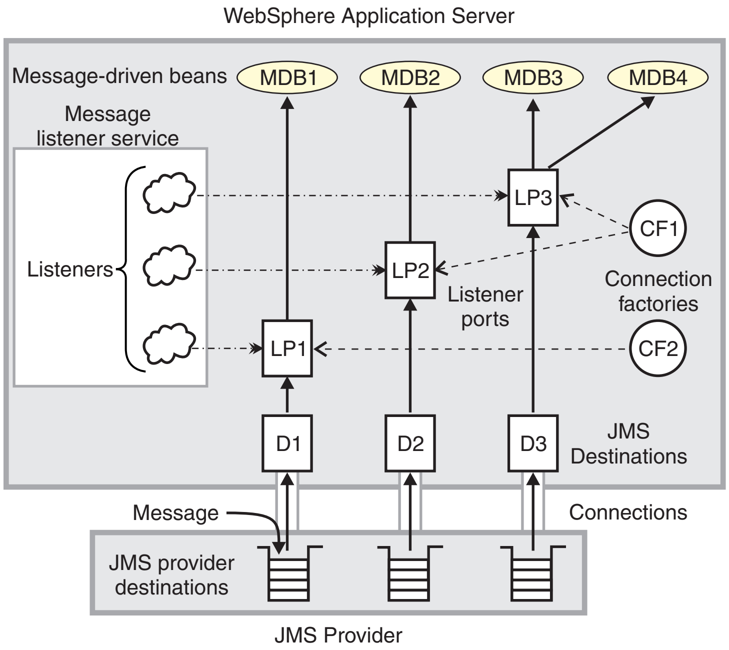


Figure 64. The main components for message-driven beans

The message listener service is an extension to the JMS functions of the JMS provider and provides a listener manager, which controls and monitors one or more JMS listeners. Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A *connection factory* is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A listener port defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports are used to simplify the administration of the associations between these resources.

When you deploy a message-driven bean, you associate the bean with a listener port. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the message listener service based on the configuration data. The message listener service creates a dynamic session thread pool for use by listeners, creates and starts listeners, and during server termination controls the cleanup of message listener service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

JMS interfaces - explicit polling for messages

Applications can use JMS to explicitly poll for messages on a destination, then retrieve messages for processing by business logic beans (enterprise beans).

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages.

The base support for asynchronous messaging that uses JMS, shown in the following figure, provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This enables WebSphere J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

Applications can use both point-to-point and publish/subscribe messaging (referred to as “messaging domains” in the JMS specification), and support the different semantics of each domain.

WebSphere Application Server supports applications that use JMS 1.1 domain-independent interfaces (referred to as the “common interfaces” in the JMS specification). With JMS 1.1, the preferred approach for implementing applications is to use the common interfaces. The JMS 1.1 common interfaces provide a simpler programming model than domain-specific interfaces. Also, applications can create both queues and topics in the same session and coordinate their use in the same transaction.

The common interfaces are also parents of domain-specific interfaces. These domain-specific interfaces (provided for JMS 1.0.2 in WebSphere Application Server Version 5) are supported only to provide inter-operation and compatibility with applications that have already been implemented to use those interfaces.

A WebSphere application can use the JMS interfaces to explicitly poll a JMS destination to retrieve an incoming message, then pass the message to a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination.

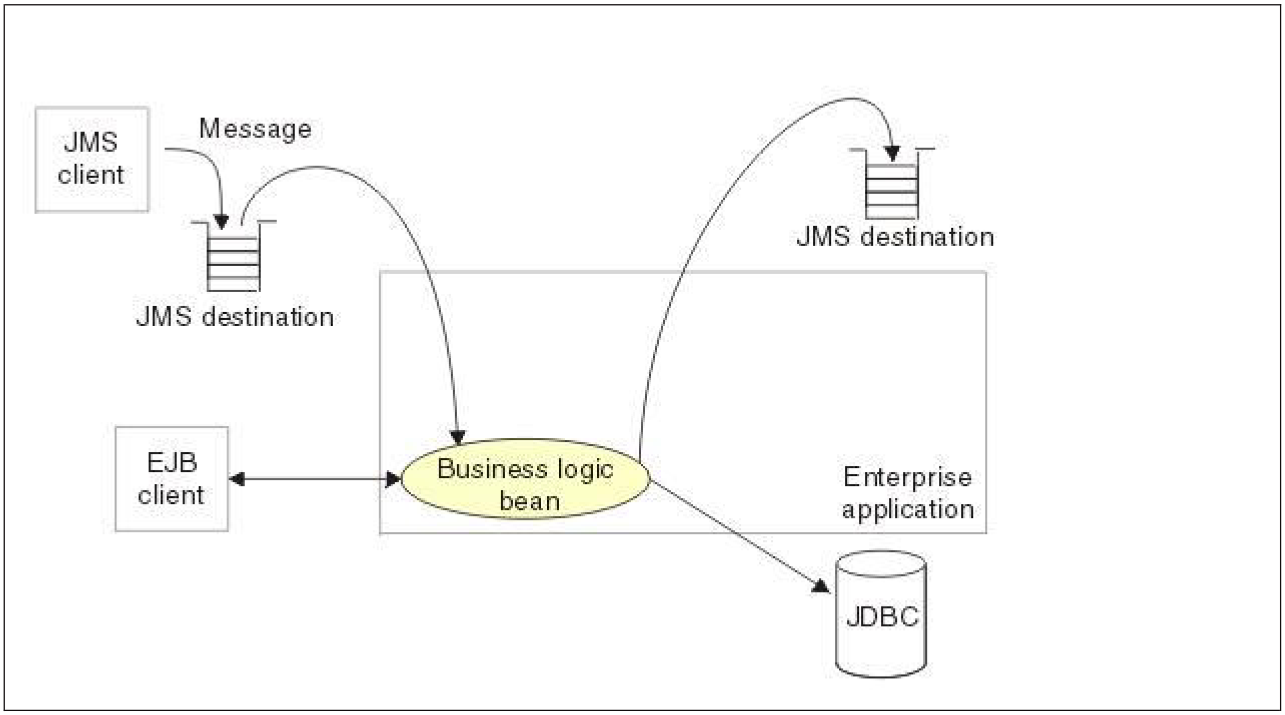


Figure 65. Asynchronous messaging by using JMS

WebSphere applications can use standard JMS calls to process messages, including any responses or outbound messaging. Responses can be handled by an enterprise bean acting as a sender bean, or handled in the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction.

WebSphere applications can also use message-driven beans, as described in the related topics about message-driven beans.

For more details about JMS, see Sun's Java Message Service (JMS) specification documentation.

Chapter 16. Naming and directory

This page provides a starting point for finding information about naming support. Naming includes both server-side and client-side components. The server-side component is a Common Object Request Broker Architecture (CORBA) naming service (CosNaming). The client-side component is a Java™ Naming and Directory Interface (JNDI) service provider. JNDI is a core component in the Java Platform, Enterprise Edition (Java EE) programming model.

The WebSphere® JNDI service provider can be used to interoperate with any CosNaming name server implementation. Yet WebSphere name servers implement an extension to CosNaming, and the JNDI service provider uses those WebSphere extensions to provide greater capability than CosNaming alone. Some added capabilities are binding and looking up of non-CORBA objects.

Java EE applications use the JNDI service provider supported by WebSphere Application Server to obtain references to objects related to server applications, such as enterprise bean (EJB) homes, which have been bound into a CosNaming name space.

Naming

Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as enterprise bean (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a *namespace*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an *initial context*. You can view the initial context as a starting point in the namespace.

The namespace structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name `myApp/myEJB` consists of one non-leaf binding with the name `myApp`, which is a context. The name also includes one leaf binding with the name `myEJB`, relative to `myApp`. The object bound with the name `myEJB` in this example happens to be an EJB home reference. The whole name `myApp/myEJB` is relative to the initial context, which you can view as a starting place when performing naming operations.

You can access and manipulate the namespace through a *name server*. Users of a name server are referred to as *naming clients*. Naming clients typically use the Java Naming and Directory Interface (JNDI) to perform naming operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.

You can use security to control access to the namespace. For more information, see *Naming roles*.

Typically, objects bound to the namespace are resources and objects associated with installed applications. These objects are bound by the system, and client applications perform lookup operations to obtain references to them. Occasionally, server and client applications bind objects to the namespace. An application can bind objects to transient or persistent partitions, depending on requirements.

In Java Platform, Enterprise Edition (Java EE) or Java Platform, Standard Edition (Java SE) environments, some JNDI operations are performed with `java:` URL names. Names bound under these names are bound to a completely different namespace which is local to the calling process. However, some lookups on the `java:` namespace may trigger indirect lookups to the name server.

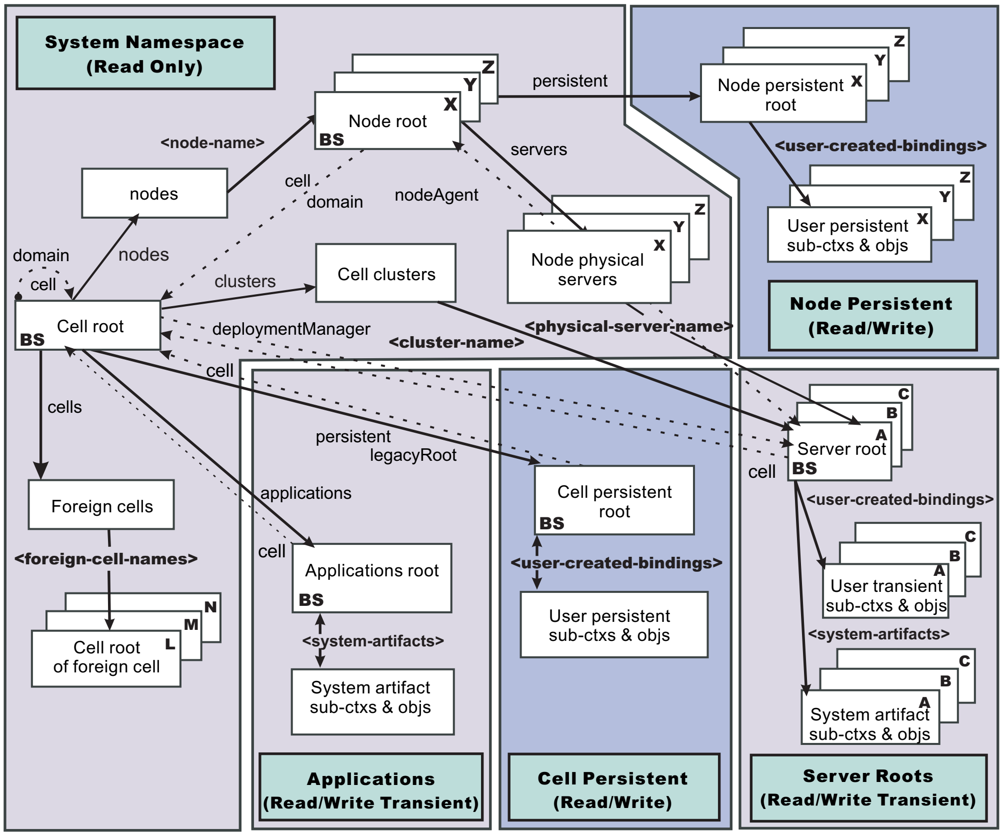
Namespace logical view

The namespace for the entire cell is federated among all servers in the cell. Every server process contains a name server. All name servers provide the same logical view of the cell namespace.

The various server roots and persistent partitions of the namespace are interconnected by a system namespace. You can use the system namespace structure to traverse to any context in a cell's namespace.

A logical view of the namespace in a multiple-server installation is shown in the following diagram.

Logical View of a Cell's Namespace



The bindings in the preceding diagram appear with solid arrows, labeled in bold, and dashed arrows, labeled in gray. Solid arrows represent *primary bindings*. A primary binding is formed when the associated subcontext is created. Dashed arrows show *linked bindings*. A linked binding is formed when an existing context is bound under an additional name. Linked bindings are added for convenience or interoperability with previous WebSphere Application Server versions.

A cell namespace is composed of contexts which reside in servers throughout the cell. All name servers in the cell provide the same logical view of the cell namespace. A name server constructs this view at startup by reading configuration information. Each name server has its own local in-memory copy of the namespace and does not require another running server to function. There are, however, a few exceptions. Server roots for other servers are not replicated among all the servers. The respective server for a server root must be running to access that server root context.

Namespace partitions

There are five major partitions in a cell namespace:
- System namespace partition
- Server roots partition
- Cell persistent partition
- Node persistent partition
- Applications partition

System namespace partition

The system namespace contains a structure of contexts based on the cell topology. The system structure supports traversal to all parts of a cell namespace and to the cell root of other cells, which are configured as foreign cells. The root of this structure is the cell root. In addition to the cell root, the system structure contains a node root for each node in the cell. You can access other contexts of interest specific to a node from the node root, such as the node persistent root and server roots for servers configured in that node.

All contexts in the system namespace are read-only. You cannot add, update, or remove any bindings.

Server roots partition

Each server in a cell has a server root context. A server root is specific to a particular server. You can view the server roots for all servers in a cell as being in a transient read/write partition of the cell namespace. System artifacts, such as enterprise bean (EJB) homes for server applications and resources, are bound under the server root context of the associated server. A server application can also add bindings under its server root. These bindings are transient. Therefore, the server application creates all required bindings at application startup, so they exist anytime the application is running.

Server-scoped configured name bindings are relative to a server's server root.

Cell persistent partition

The root context of the cell persistent partition is the cell persistent root. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed. Applications that need to create additional persistent bindings of objects generally associated with the cell can bind these objects under the cell persistent root.

It is important to note that the cell persistent area is not designed for transient, rapidly changing bindings. The bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Cell-scoped configured name bindings are relative to a cell's cell persistent root.

Node persistent partition

The node persistent partition is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent bindings of objects associated with a specific node can bind those objects under that particular node's node persistent root. As with the cell persistent area, it is important to note that the node persistent area is not designed for transient, rapidly changing bindings. These bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Node-scoped configured name bindings are relative to a node's node persistent root.

Applications partition

The Java EE 6 specification introduces module, application, and global namespaces. Java URL JNDI names that have the prefixes `java:module`, `java:app`, and `java:global` can access the respective namespaces. In some situations, the namespaces are only locally accessible, and in other situations the namespaces are remotely accessible.

The applications partition contains namespaces that are remotely accessible. The root of the `java:global` namespace is the applications root context. The roots of other namespaces are under the com.ibm.ws.AppNameSpaces subcontext. For example, the `java:app` root context for the application, MyApp, is bound with the name, `MyApp/root` relative to com.ibm.ws.AppNameSpaces. Module and component namespaces are only accessible remotely when the module is a client module in a server-deployed mode or in a federated mode. For example, the `java:module` root context for the server-deployed client module MyClientModule in the application MyApp is bound with the name `MyApp/MyClientModule/root` relative to com.ibm.ws.AppNameSpaces. The component namespace, which contains `comp/env` bindings, for that same module is bound under `MyApp/MyClientModule/ClientComponent/root` relative to com.ibm.ws.AppNameSpaces.

Application resources–such as EJB references, resource references, and environment entries–with `java:global` names are bound into the `java:global` namespace when the defining application is installed. The application does not need to be running for those name bindings to be available to other applications.

Initial context support

All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the namespace. Use the initial context to perform naming operations, such as looking up and binding objects in the namespace.

Initial contexts registered with the ORB as initial references

The root contexts listed in the following table are registered with the name server's ORB and can be used as an initial context. An initial context is used by CORBA and enterprise bean applications as a starting point for namespace lookups. The keys for these roots as recognized by the ORB are shown in the following table:

Server Root	**NameServiceServerRoot**
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot, NameService
Node Root	NameServiceNodeRoot
Applications Root	NameServiceApplicationsRoot

A server root initial context is the server root context for the specific server you are accessing. Similarly, a node root initial context is the node root for the server being accessed.

You can use the previously mentioned keys in CORBA INS object URLs (corbaloc and corbaname) and as an argument to an ORB `resolve_initial_references` call. For examples, see CORBA and JNDI programming examples, which show how to get an initial context.

Default initial contexts

The default initial context depends on the type of client. Different categories of clients and the corresponding default initial context follow.

WebSphere Application Server JNDI interface implementation

The JNDI interface is used by EJB applications to perform namespace lookups. WebSphere Application Server clients by default use the WebSphere Application Server CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the server root of the server specified by the provider URL. For more details, refer to the JNDI programming examples on getting initial contexts.

Other JNDI implementation

Some applications can perform namespace lookups with a non-product CosNaming JNDI plug-in implementation. Assuming the key `NameService` is used to obtain the initial context, the default initial context for clients of this type is the cell root.

CORBA

The standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key `NameService`. The initial context in this case is the cell root.

Lookup names support in deployment descriptors and thin clients

Server application objects, such as enterprise bean (EJB) homes, are bound relative to the server root context for the server in which the application is installed. Other objects, such as resources, can also be bound to a specific server root. The names used to look up these objects must be qualified so as to select the correct server root. This topic discusses what relative and qualified names are, when they can be used, and how you can construct them.

Beginning in Version 8.0, EJB homes are bound under the name, `java:global/`*`appName`*`/`*`moduleName`*`/`*`beanName`*. Names of that form are not topology-based and are fully-qualified already. Similarly, all application resources that are bound with `java:global`, `java:app`, or `java:module` names need no additional qualification when the `java:global`, `java:app`, or `java:module` lookup name is specified. Application resources include, for example, EJB references, resource references, and environment entries.

Relative names

All names are relative to a context. Therefore, a name that can be resolved from one context in the namespace cannot necessarily be resolved from another context in the namespace. This point is significant because the system binds objects with names relative to the server root context of the server in which the application is installed. Each server has its own server root context. The initial Java Naming and Directory Interface (JNDI) context is by default the server root context for the server identified by the provider URL used to obtain the initial context. (Typically, the URL consists of a host and port.) For applications running in a server process, the default initial JNDI context is the server root for that server. A relative name will resolve successfully when the initial context is obtained from the server which contains the target object, but it will not resolve successfully from an initial context obtained from another server.

If all clients of a server application run in the same server process as the application, all objects associated with that application are bound to the same initial context as the clients' initial context. In this case, only names relative to the server's server root context are required to access these server objects. Frequently, however, a server application has clients that run outside the application's server process. The initial context for these clients can be different from the server application's initial context, and lookups on the relative names for server objects may fail. These clients need to use the qualified name for the server objects. This point must be considered when setting up the jndiName values in a Java Platform, Enterprise Edition (Java EE) client application deployment descriptors and when constructing lookup names in thin clients. Qualified names resolve successfully from any initial context in the cell.

Qualified names

All names are relative to a context. Here, the term *qualified name* refers to names that can be resolved from any initial context in a cell. This action is accomplished by using names that navigate to the same context, the cell root. The rest of the qualified name is then relative to the cell root and uniquely identifies an object throughout the cell. All initial contexts in a server (that is, all naming contexts in a server registered with the ORB as an initial reference) contain a binding with the name `cell`, which links back to the cell root context. All qualified names begin with the string `cell/` to navigate from the current initial context back to the cell root context.

A qualified name for an object is the same throughout the cell. The name can be topology-based, or some fixed name bound under the cell persistent root. Topology-based names, described in more detail later in this section, navigate through the system namespace to reach the target object. A fixed name bound under the cell persistent root has the same qualified name throughout the cell and is independent of the topology. Creating a fixed name under the cell persistent root for a server application object requires an extra step when the server application is installed, but this step eliminates impacts to clients when the application is moved to a different location in the cell topology. The process for creating a fixed name is also described later.

Generally, you must use qualified names for EJB jndiName values in a Java EE client application deployment descriptors and for EJB lookup names in thin clients. The only exception is when the initial context is obtained from the server in which the target object resides. For example, a session bean which is a client to an entity bean can use a relative name if the two beans run in the same server. If the session bean and entity beans run in different servers, the jndiName for the entity bean must be qualified in the session bean's deployment descriptors. The same requirement may be true for resources as well, depending on the scope of the resource.

- **Topology-based names**

  The system namespace partition in a cell's namespace reflects the cell's topology. This structure can be navigated to reach any object bound into the cell's namespace. Topology-based qualified names include elements from the topology which reflect the object's location within the cell.

  Single server

  An object bound in a single server has a topology-based qualified name of the following form:

  ```
  cell/nodes/nodeName/servers/serverName/relativeJndiName
  ```

  where *nodeName* and *serverName* are the node name and server name for the server where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to its server's server root context.

- **Fixed names**

  It is possible to create a fixed name for a server object so that the qualified name is independent of the cell topology. This quality is desirable when clients of the application run in other server processes or as pure clients. Fixed names have the advantage of not changing if the object is moved to another server. The jndiName values in deployment descriptors for a Java EE client application can reference the qualified fixed name for a server object regardless of the cell topology on which the client or server application is being installed.

  Defining a cell-wide fixed name for a server application object requires an extra step after the server application is installed. That is, a binding for the object must be created under the cell persistent root. A fixed name bound under the cell persistent root can be any name, but all names under the cell persistent root must be unique within the cell because the cell persistent root is global to the entire cell.

  A qualified fixed name has the form:

  ```
  cell/persistent/fixedName
  ```

  where *fixedName* is an arbitrary fixed name.

  The binding can be created programmatically (for example, using JNDI). However, it is probably more convenient to configure a cell-scoped binding for the server object.

  You must keep the programmatic or configured binding up-to-date. Configured EJB bindings are based on the location of the enterprise bean within the cell topology, and moving the EJB application to another server, for example, requires the configured binding to be updated. Similar changes affect an EJB home reference programmatically bound so that the fixed name would need to be rebound with a current reference. However, for Java EE clients, the jndiName value for the object, and for thin clients, the lookup name for the object, remains the same. In other words, clients that access objects by fixed names are not affected by changes to the configuration of server applications they access.

Using lookup names in deployment descriptor bindings

Java EE applications can contain deployment descriptors, such as ejb-ref, resource-ref, and resource-env-ref, that are used to declare various types of references. These reference declarations define `java:comp/env` lookup names that are available to corresponding Java EE components. Each `java:comp/env` lookup name must be mapped to a lookup name in the global name space, relative to the server root context, which is the default initial JNDI context.

If a reference maps to an object that is bound under the server root context for the same server as the component that is executing the lookup, you can use a relative lookup name. If a reference maps to an object that is bound under the server root context of another server, you must qualify the lookup name. For example, you must qualify a lookup name if a servlet, that is running on one server, declares an ejb-ref for an EJB that is running on another server. Similarly, if the reference maps to an object that is bound into a persistent partition of the name space, or to an object that is bound through a cell-scoped or node-scoped configured name space binding, you must use a qualified name.

You can specify deployment descriptor reference binding values when you install the application, and edit them after the application is installed. If you need to change the JNDI lookup name a reference maps to, in the administrative console, click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***. In the References section, there are links that correspond to the various reference types, such as EJB references and Resource environment entry references, that are declared by this application. Click on the link for the reference type that you need to change, and then specify a new value in the **Target Resource JNDI Name** field.

JNDI support in WebSphere Application Server

The product includes a name server to provide shared access to Java components, and an implementation of the `javax.naming` JNDI package which supports user access to the name server through the Java Naming and Directory Interface (JNDI) naming interface.

The product does not provide implementations for:
- `javax.naming.directory` or
- `javax.naming.ldap` packages

Also, the product does not support interfaces defined in the `javax.naming.event` package.

However, to provide access to LDAP servers, the development kit shipped with the product supports the implementation of:
- `javax.naming.ldap` and
- `com.sun.jndi.ldap.LdapCtxFactory`

The WebSphere Application Server JNDI implementation is based on the JNDI interface, and was tested with the JNDI Service Provider Interface (SPI).

The default behavior of this JNDI implementation is adequate for most users. However, users with specific requirements can control certain aspects of JNDI behavior.

Configured name bindings

Administrators can configure bindings into the namespace. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

Administrators can add name bindings to the namespace through the configuration. Name servers add these configured bindings to the namespace view, by reading the configuration data for the bindings. Configuring bindings is an alternative to creating the bindings from a program. Configured bindings have the advantage of being created each time a server starts, even when the binding is created in a transient partition of the namespace. Cell-scoped configured bindings provide a fixed qualified name for server application objects.

Scope

You can configure a binding at one of the following four scopes: cell, node, server, or cluster. Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server. Cluster-scoped bindings are created under the server root context in each member of the selected cluster.

The scope you select for new bindings depends on how the binding is to be used. For example, if the binding is not specific to any particular node, cluster, or server, or if you do not want the binding to be associated with any specific node, cluster, or server, a cell-scoped binding is a suitable scope. Defining fixed names for enterprise beans to create fixed qualified names is just such an application. If a binding is to be used only by clients of an application running on a particular server (or cluster), or if you want to configure a binding with the same name on different servers (or clusters) which resolve to different objects, a server-scoped (or cluster-scoped) binding would be appropriate. Note that two servers or clusters can have configured bindings with the same name but resolve to different objects. At the cell scope, only one binding with a given name can exist.

Intermediate contexts

Intermediate contexts created with configured bindings are read-only. For example, if an EJB home binding is configured with the name `some/compound/name/ejbHome`, the intermediate contexts `some`, `some/compound`, and `some/compound/name` will be created as read-only contexts. You cannot add, update, or remove any read-only bindings.

The configured binding name cannot conflict with existing bindings. However, configured bindings can use the same intermediate context names. Therefore, a configured binding with the name `some/compound/name2/ejbHome2` does not conflict with the previous example name.

Configured binding types

Types of objects that you can bind follow:

EJB: EJB home installed in some server in the cell

The following data is required to configure an EJB home binding:
- JNDI name of the EJB server or server cluster where the enterprise bean is deployed
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped EJB binding is useful for creating a fixed lookup name for an enterprise bean so that the qualified name is not dependent on the topology.

Note: In stand-alone servers, an EJB binding resolving to another server cannot be configured because the name server does not read configuration data for other servers. That data is required to construct the binding.

CORBA: CORBA object available from some CosNaming name server

You can identify any CORBA object bound into some INS compliant CosNaming server with a corbaname URL. The referenced object does not have to be available until the binding is actually referenced by some application.

The following data is required in order to configure a CORBA object binding:
- The corbaname URL of the CORBA object
- An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object)
- Target root for the configured binding
- The name of the configured binding, relative to the target root

Indirect: Any object bound in WebSphere Application Server namespace accessible with JNDI

Besides CORBA objects, this includes javax.naming.Referenceable, javax.naming.Reference, and java.io.Serializable objects. The target object itself is not bound to the namespace. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. The following data is required in order to configure an indirect JNDI lookup binding:
- JNDI provider URL of name server where object resides
- JNDI lookup name of object

- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped indirect binding is useful when creating a fixed lookup name for a resource so that the qualified name is not dependent on the topology. You can also achieve this topology by widening the scope of the resource definition.

String: String constant

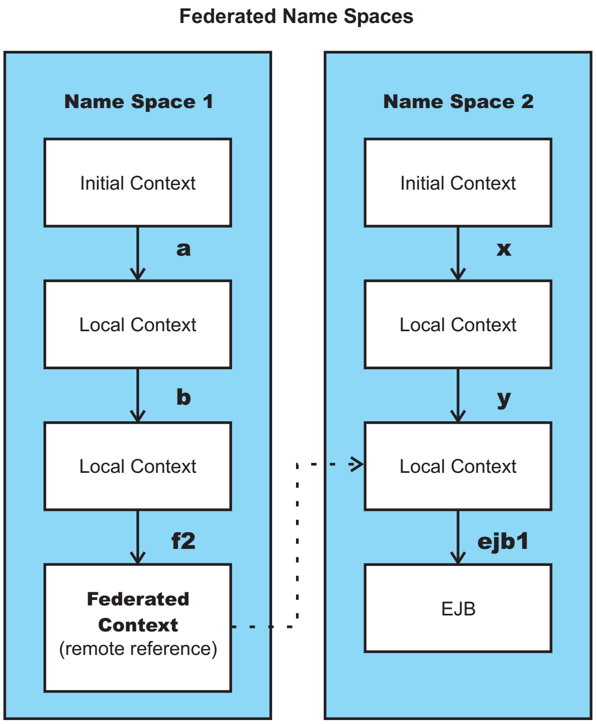
You can configure a binding of a string constant. The following data is required to configure a string constant binding:

- String constant value
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root

Namespace federation

Federating namespaces involves binding contexts from one namespace into another namespace.

For example, assume that a namespace, Namespace 1, contains a context under the name `a/b`. Also assume that a second namespace, Namespace 2, contains a context under the name `x/y`. (See the following illustration.) If context `x/y` in Namespace 2 is bound into context `a/b` in Namespace 1 under the name `f2`, the two namespaces are federated. Binding `f2` is a federated binding because the context associated with that binding comes from another namespace. From Namespace 1, a lookup of the name `a/b/f2` returns the context bound under the name `x/y` in Namespace 2. Furthermore, if context `x/y` contains an enterprise bean (EJB) home bound under the name `ejb1`, the EJB home can be looked up from Namespace 1 with the lookup name `a/b/f2/ejb1`. Notice that the name crosses namespaces. This fact is transparent to the naming client.



In a product namespace, you can create federated bindings with the following restrictions:

- Federation is limited to CosNaming name servers. A product name server is a Common Object Request Broker Architecture (CORBA) CosNaming implementation. You can create federated bindings to other CosNaming contexts. You cannot, for example, bind contexts from an LDAP name server implementation.
- If you use JNDI to federate the namespace, you must use a WebSphere Application Server initial context factory to obtain the reference to the federated context. If you use some other initial context factory implementation, you might not be able to create the binding or the level of transparency might be reduced.
- A federated binding to a non-product naming context has the following functional limitations:
  - JNDI operations are restricted to the use of CORBA objects. For example, you can look up EJB homes, but you cannot look up non-CORBA objects such as data sources.
  - JNDI caching is not supported for non-product namespaces. This restriction affects the performance of lookup operations only.
  - If security is enabled, the producr does not support federated bindings to non-product namespaces.
- Do not federate two product stand-alone server namespaces. Incorrect behavior might result. If you want to federate product namespaces, use servers running under the WebSphere Application Server, Network Deployment package of WebSphere Application Server.
- When federating the namespaces of two cells running a WebSphere Application Server, Network Deployment package of WebSphere Application Server, the names of the cells must be different. Otherwise, incorrect behavior can result.

Naming roles

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the name space. Generally two ways are acceptable in which client programs result in CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) methods. The second is CORBA clients invoking CosNaming methods directly.

The following security roles exist. However, the roles have an authority level from low to high as shown in the following list. The list also provides the security-related interface methods for each role. The interface methods that are not listed are either not supported or not relevant to security.

- CosNamingRead. Users who are assigned the CosNamingRead role can do queries of the name space, such as through the JNDI lookup method. The Everyone special-subject is the default policy for this role.

Table 36. CosNamingRead role packages and interface methods. The following table lists the CosNamingRead role packages and interface methods:

Package	Interface methods
javax.naming	• Context.list • Context.listBindings • Context.lookup • NamingEnumeration.hasMore • NamingEnumeration.next
org.omg.CosNaming	• NamingContext.list • NamingContext.resolve • BindingIterator.next_one • BindingIterator.next_n • BindingIterator.destroy

- CosNamingWrite. Users who are assigned the CosNamingWrite role can do write operations (such as JNDI bind, rebind, or unbind) plus CosNamingRead operations. As a default policy, Subjects are not assigned this role.

Table 37. CosNamingWrite role packages and interface methods. The following table lists the CosNamingWrite role packages and interface methods:

Package	Interface methods
javax.naming	• Context.bind • Context.rebind • Context.rename • Context.unbind
org.omg.CosNaming	• NamingContext.bind • NamingContext.bind_context • NamingContext.rebind • NamingContext.rebind_context • NamingContext.unbind

- CosNamingCreate. Users who are assigned the CosNamingCreate role are allowed to create new objects in the name space through JNDI createSubcontext operations plus CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

Table 38. CosNamingCreate role packages, interface methods. The following table lists the CosNamingCreate role packages, interface methods:

Package	Interface methods
javax.naming	Context.createSubcontext
org.omg.CosNaming	NamingContext.bind_new_context

- CosNamingDelete. Users who are assigned the CosNamingDelete role can destroy objects in the name space, for example byusing the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

Table 39. CosNamingDelete role packages and interface methods. The following table lists the CosNamingDelete role packages and interface methods:

Package	Interface methods
javax.naming	Context.destroySubcontext
org.omg.CosNaming	NamingContext.destroy

Important: The `javax.naming` package applies to the CosNaming JNDI service provider only. All of the variants of a JNDI interface method have the same role mapping.

If the caller is not authorized, the packages listed in the previous tables exhibit the following behavior:

javax.naming
This package creates the javax.naming.NoPermissionException exception, which maps NO_PERMISSION from the CosNaming method invocation to NoPermissionException.

org.omg.CosNaming
This package creates the org.omg.CORBA.NO_PERMISSION exception.

Users, groups, or the AllAuthenticated and Everyone special subjects can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that is assigned a different naming role, the user is granted the most permissive access between the role that is assigned and the role the group is assigned. For example, assume that the MyUser user is assigned the CosNamingRead role. Also, assume that the MyGroup group is assigned the CosNamingCreate role. If the

MyUser user is a member of the MyGroup group, the MyUser user is assigned the CosNamingCreate role because the user is a member of the MyGroup group. If the MyUser user is not a member of the MyGroup group, is assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when administrative security is enabled. When administrative security is enabled, attempts to do CosNaming operations without the proper role assignment result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

In WebSphere Application Server, each CosNaming function is assigned to one role only. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they also are assigned the CosNamingRead role. In most cases, a creator needs three roles assigned: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator in the previous example have been included in CosNamingCreate role. In most cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, doing so might result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime. Typically, J2EE applications access the name space and the identity is that of the user that authenticated to WebSphere Application Server when he J2EE application is accessed. Unless the J2EE application provider clearly communicates the expected naming roles, fully consider changing the default naming authorization policy.

Naming and directories: Resources for learning

Additional information and guidance on naming and directories is available on various Internet sites.

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

The naming service provided with WebSphere Application Server Versions 6.x, 7.x, and 8.0 is the same as that provided for Version 5, thus information on the Version 5.0 naming and directories applies to Version 6.0 and later.

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

Programming instructions and examples

- Naming in WebSphere Application Server V5: Impact on Migration and Interoperability, `http://www.ibm.com/developerworks/websphere/library/techarticles/0305_weiner/weiner.html`
- *WebSphere Application Server V6.1:* System Management Configuration Handbook, SG24-7304-00, `http://www.redbooks.ibm.com/abstracts/SG247304.html?Open`
- *IBM WebSphere Developer Technical Journal:* Co-hosting multiple versions of J2EE applications, `http://www.ibm.com/developerworks/websphere/techjournal/0405_poddar/0405_poddar.html`

Programming specifications

- Specifications and API documentation

Chapter 17. Object Request Broker (ORB)

This page provides a starting point for finding information about the Object Request Broker (ORB). The product uses an ORB to manage communication between client applications and server applications as well as among product components. These Java Platform, Enterprise Edition (Java EE) standard services are relevant to the ORB: Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP) and Java Interface Definition Language (Java IDL).

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency.

Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and to call operations on those objects as if the remote objects are located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a *stub*. The stub forwards the request to the remote object, where the operation runs and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and required parameters, and for sending the request on the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which, in turn, returns to the client application, as if the operation had been run locally.

This product uses an ORB to manage communication between client applications and server applications as well as communication among product components. During product installation, default property values are set when the ORB is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. This product does not support the use of multiple ORB instances.

Logical pool distribution

The Logical pool distribution (LPD) thread pool mechanism implements a strategy for improving the performance of requests that have shorter run times. Do not configure LPD unless you have already configured it in a previous release of the product.

depfeat: LPD is a deprecated function and will be removed in a future version of the product.

The need for LPD is indicated by a mixture of Enterprise JavaBeans (EJB) requests where the run times vary across the request types, and the ORB thread pool must be constrained for performance reasons. In this case, longer run time requests might tend to prolong the response times for shorter requests by denying them adequate access to threads in the thread pool. LPD provides a mechanism that allows shorter requests greater access to the threads.

LPD divides the Object Request Broker (ORB) thread pool into logical pools, as configured by the administrator using ORB custom properties starting that start with the following:

```
com.ibm.websphere.threadpool.strategy.*
```

The size of each pool is a percentage of the maximum number of ORB threads. The sum of the logical pool percentages must equal 100.

When LPD is active, incoming ORB requests are vectored, or pointed, to a pool based on historical run time history for the request type. The request type is determined by the method, which is qualified internally as unique across components. The LPD mechanism adjusts pool targets at runtime to optimize the distribution of requests across logical pools.

The LPD mechanism can be tuned after it is enabled. Response time, throughput measurements, and statistics produced by the LPD mechanism drive the tuning process.

Object Request Brokers: Resources for learning

Use the following links to find relevant supplemental information about Object Request Brokers (ORBs). The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

View links to additional information about:
- “Planning, business scenarios, and IT architecture”
- “Administration”
- “Programming specifications”

Planning, business scenarios, and IT architecture

- CORBA FAQ

  Getting started with Object Request Brokers and CORBA.
- WebSphere Application Server CORBA Interoperability

  This document describes WebSphere CORBA interoperability for WebSphere Application Server products.
- CORBA Interoperability Samples

  These samples demonstrate the general principles by which WebSphere Application Server applications can interoperate with CORBA applications.

Administration

- IANA Character Set Registry

  This document contains a list of all valid character encoding schemes.
- developerWorks WebSphere

Programming specifications

- Catalog Of OMG CORBA/IIOP Specifications

  This document provides a catalog of OMG CORBA/IIOP specifications.

Chapter 18. OSGi applications

The OSGi applications support in WebSphere Application Server helps you develop and deploy modular applications that use both Java EE and OSGi technologies. You can design and build applications and suites of applications from coherent, versioned, reusable OSGi modules that are accessed only through well-defined interfaces. This enables the same, or different, applications to use different versions of the same third party libraries without interference.

Apache Aries is an open community project that brings the modularity, dynamism, and versioning of the OSGi service platform to enterprise application developers by implementing key EEG specifications. Apache Aries delivers a simple to use and lightweight programming model for web applications that combines the standard Blueprint component model with familiar Java enterprise technologies. Apache Aries includes an implementation of the OSGi service platform Version 4.2 Blueprint component model for fine-grained assembly, and provides an assembly model for applications that consist of multiple modules.

The OSGi applications support in WebSphere Application Server includes the following major features:

- Use the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification Blueprint Container for declarative assembly of components. This simplifies unit test outside of the application server.
- Use extensions to the Blueprint component model for declarative transactions and container-managed Java Persistence API (JPA).
- Develop OSGi application projects using IBM Rational Application Developer, which enforces OSGi visibility rules so that projects can only access packages from other projects that explicitly declare them as part of the project externals. This provides environmental support to development best practices.
- Compose isolated enterprise applications using multiple, versioned bundles with dynamic life cycle.
- Deploy applications in archive files that contain only application-specific content and metadata that points to shared bundles. This means that application archive files can be smaller. It also means that, when a library is shared by several OSGi applications, only one copy of the library is loaded into memory.
- Use an integrated bundle repository, and configure the locations of external repositories, to support the provisioning of bundles to applications.
- Deploy existing web application archive (WAR) files as web application bundles (WABs). This allows web applications to use the OSGi module system.
- Deploy existing EJB JAR files as EJB bundles.
- Deploy web applications that use Version 3.0 of the Java Servlet Specification.
- Deploy enterprise applications that contain EJB 3.x style enterprise beans.
- Simultaneously load multiple versions of classes in the same application, using standard OSGi mechanisms.
- Administratively update deployed applications in a modular fashion, at the bundle-level.
- Deploy applications that use their own versions of common utility classes, distinct from the versions that are used by the server runtime environment. Do this without needing to configure application Java EE class loader policies, such as PARENT_LAST mode.
- Use federated lookup mechanisms between the local Java Naming and Directory Interface (JNDI) and the OSGi service registry.
- Extend and scale running applications, as business needs change, without changing the underlying application.
- Update a running application, only impacting those bundles affected by the update.

An introduction to OSGi Applications

The OSGi Applications feature of WebSphere Application Server integrates Apache Aries technologies, including the Blueprint Container and OSGi application assembly model, into WebSphere Application Server. The integration of Apache Aries into WebSphere Application Server addresses many of the challenges of developing and maintaining extensible web applications.

Using OSGi Applications, you can deploy and manage your web applications as a set of versioned OSGi bundles. You can also configure one or more bundle repositories, as part of the provisioning infrastructure, to host common bundles used by multiple applications, and to simplify the deployment of applications that use those common bundles.

Presentations:

- Modular and dynamic OSGi applications: Part 1: Motivations and specifications (22 min)
- Modular and dynamic OSGi applications: Part 2: OSGi applications support in WebSphere Application Server (26 min)

Business goals and OSGi Applications

The OSGi Applications support in WebSphere Application Server brings the modularity, dynamism, and versioning of the OSGi service platform to enterprise web application developers. This reduces complexity, and provides the greatest flexibility to maintain and evolve an application after its first release. You can use OSGi Applications to combine the standard Blueprint component model with familiar Java enterprise technologies.

OSGi Applications support is focused on the web-based technologies that many applications use. This includes the Spring Dynamic Modules project, which many web applications use for fine-grained component assembly and management, and which inspired the OSGi Blueprint component model. WebSphere Application Server provides an implementation of the OSGi Blueprint Container that was developed in the Apache Aries project. Applications that are composed from Blueprint components can rely on the Blueprint Container that the application server runtime environment provides, in contrast to Spring-based applications, which include the Spring container as part of the application itself.

OSGi modularity provides standard mechanisms to address common challenges with enterprise Java applications. The OSGi Applications support in WebSphere Application Server provides the following major benefits:

- It helps your applications to be more portable, easier to re-engineer, and more adaptable to changing requirements.
- It provides the declarative assembly and simplified unit test of dependency injection frameworks such as the Spring Framework, but in a standardized and IBM-supported form that is provided as part of the application server run time rather than being a third-party library deployed as part of the application.
- It integrates fully with the Java EE programming model, giving you the option of deploying web modules and EJB modules as versioned OSGi bundles with dynamic life cycles.
- It supports administration of application bundle dependencies and versions, which simplifies and standardizes third-party library integration.
- It provides isolation for enterprise applications that are composed of multiple, versioned bundles with dynamic life cycles.
- It has a built-in bundle repository that can host common and versioned bundles shared between multiple applications, so that each application does not deploy its own copy of each common library.
- It can access external bundle repositories.
- It reinforces service-oriented design at the module level.
- It composes into coarser-grained Service Component Architecture (SCA) assemblies.

When you use the OSGi Applications support in WebSphere Application Server, you are using a standards-based programming model, and also gaining the well-understood benefits of WebSphere Application Server administration, performance and enterprise-level qualities of service.

The modularization challenge

OSGi is a dynamic module system for Java. So how does it help?

Effective software modules have the following characteristics:

- **Self-contained**: Although a module is comprised of smaller parts, it is the whole module that can be moved around, installed, or uninstalled as a single unit, not the parts within it.
- **Highly cohesive**: Each module has a coherent logical function.
- **Loosely-coupled**: Modules have well-defined boundaries between them.

Modularized systems that have these characteristics are easier to maintain and extend.

Object-oriented languages such as Java support modularization. However, they focus on encapsulation of instance variables. This helps at the object and class level, but does not support higher forms of modularity. Java EE helps a little more by providing application-level isolation of application modules within an enterprise application.

Patterns such as SOA and Dependency Injection encourage modular design of large-scale enterprise applications. However, this modularity requires architectural governance rather than being encouraged or enforced by the runtime environment.

In the Java platform, data is encapsulated within a class, classes are scoped within a package, and packages are collected together in a Java archive (JAR) file. Java class visibility options are *private*, *package*, *protected*, and *public*. There is no access modifier that allows for a unit of deployment that is a JAR file rather than a package. Most JAR files consist of multiple packages, and if the JAR file represents a cohesive function, there is typically a need for classes in one package to access classes in another package in the same JAR file. This need requires *public* accessibility of that class, which also makes the class visible to classes in other JAR files. JAR files provide no visibility control. Even well-behaved applications that use only the classes a JAR file provider expects to be used externally are governed by the Java class path, because a required class might be available from multiple JAR files and the class that is loaded is the first available instance on the global class path.

JAR files cannot scope the visibility of what they contain, and also cannot declare their own dependencies. Many JAR files have implicit dependencies on other JAR files, which means these JAR files cannot be installed or moved around independently. If a JAR file is installed and its dependencies are missing, the problem is often not visible until run time.

Java class loading scans the class path to look inside each JAR file on the class path to locate the required class. This process has three main limitations:

- Class path ordering determines which instance of a class is loaded, and therefore which JAR file it is loaded from.
- Only one version of a class is available on the class path, again determined by the first instance that is found.
- If the dependencies of a class are not resolved, the first indication of the problem is often a runtime ClassNotFoundException exception.

These class path and JAR file shortcomings are often referred to as “JAR hell”. Java EE partly mitigates these problems. Java EE introduces the enterprise archive (EAR) file, both as the method by which an enterprise application is delivered, and as a runtime isolation scope for the modules that are part of that application. Java EE applications have a class loader hierarchy that is partly shared between the enterprise applications, and partly isolated between the applications. For example, in an enterprise application that contains a web application archive (WAR) module, by default, the individual WAR modules are isolated from each other in the application, and isolated from anything in a different application.

While the “JAR hell” problems are reduced by managing different class paths with different enterprise applications, there are still limitations when you want to share libraries such as open source frameworks or utility libraries between applications. WebSphere Application Server offers some advanced options for configuring enterprise applications to access libraries that are not delivered as part of the EAR file:

- You can install an isolated library and administratively associate its classloader with one or more installed modules or applications, or associate the classloader with the server to make it visible to all application modules.
- You can configure the classloader delegation pattern to help resolve versioning compatibility problems. For example, you can specify the class loader delegation mode as parent-last so that an application-supplied class is loaded in preference to a server-supplied class.

However, these approaches only partially address the modularity requirements of applications. “The OSGi Framework” offers a better solution.

The OSGi Framework

OSGi defines a dynamic module system for Java. The OSGi service platform has a layered architecture, and is designed to run on various standard Java profiles.

OSGi Applications deployed to WebSphere Application Server run on an Enterprise Java profile that is provided as part of the server runtime environment. This environment also provides the OSGi framework in which OSGi Applications run. Eclipse Equinox is the reference implementation of the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification, and WebSphere Application Server uses Equinox as the framework for OSGi Applications. The precise version of Equinox depends on the service level of WebSphere Application Server.

The integrated OSGi framework in WebSphere Application Server provides support for each of the layers of the OSGi Architecture:

- “Modules layer”
- “Lifecycle layer” on page 479
- “Services layer” on page 479

Modules layer

The unit of deployment in OSGi is a bundle. The modules layer is where the OSGi Framework processes the modular aspects of a bundle. The metadata that enables the OSGi Framework to do this processing is provided in a bundle manifest file. For more information about this file, see “Example: OSGi bundle manifest file” on page 508.

One key advantage of OSGi is its class loader model, which uses the metadata in the manifest file. There is no global class path in OSGi. When bundles are installed into the OSGi Framework, their metadata is processed by the module layer and their declared external dependencies are reconciled against the versioned exports declared by other installed modules. The OSGi Framework works out all the dependencies, and calculates the independent required class path for each bundle. This approach resolves the shortcomings of plain Java class loading by ensuring that the following requirements are met:

- Each bundle provides visibility only to Java packages that it explicitly exports.
- Each bundle declares its package dependencies explicitly.
- Packages can be exported at specific versions, and imported at specific versions or from a specific range of versions.
- Multiple versions of a package can be available concurrently to different clients.

Lifecycle layer

The bundle lifecycle management layer in OSGi enables bundles to be dynamically installed, started, stopped, and uninstalled, independent from the lifecycle of the application server. The lifecycle layer ensures that bundles are started only if all their dependencies are resolved, reducing the occurrence of ClassNotFoundException exceptions at run time. If there are unresolved dependencies, the OSGi Framework reports them and does not start the bundle.

Each bundle can provide a bundle activator class, which is identified in the bundle manifest, that the framework calls on start and stop events. In this way, a bundle can provide special initialization and cleanup code if required, although most OSGi applications that are deployed to WebSphere Application Server should not need to do so. If a bundle needs a template bundle activator, you can use IBM Rational Application Developer Version 8.5 to generate one.

Services layer

The services layer in OSGi intrinsically supports a service-oriented architecture through its non-durable service registry component. Bundles publish services to the service registry, and other bundles can discover these services from the service registry.

These services are the primary means of collaboration between bundles. An OSGi service is implemented by using one of the following supported component models:

- A Blueprint-managed bean.
- An enterprise bean.
- A proxy, provided by Service Component Architecture (SCA), to a remote service

An OSGi service is published to the service registry under one or more Java interface names, with optional metadata stored as custom properties (name/value pairs). A discovering bundle can look up a service in the service registry by an interface name, and can potentially filter the services that are being looked up based on the custom properties.

Services are fully dynamic, and typically have the same lifecycle as the bundle that provides them. OSGi Applications in WebSphere Application Server usually interact with the OSGi service registry through a Blueprint module definition. POJO bean components that are described in the Blueprint module definition can be registered as services through a <service> element, or can have service references injected into them through a <reference> element.

Enterprise OSGi standards

OSGi for Java enterprise applications is one focus of Version 4.2 of the OSGi service platform specification, which introduces the OSGi Service Platform Enterprise Specification.

This specification includes the definition of the Blueprint component model, which is derived from the Spring Dynamic Modules project. The Blueprint component model forms an important part of the OSGi Applications programming model in WebSphere products. It describes how components can be wired together within a bundle, how components can be published as services to the service registry, and how components can have configuration and dependencies injected into them by a Blueprint component container that is part of the runtime environment.

Components and the references that they consume are declared in an XML module Blueprint file, which is a standardization of the Spring application context. This is extended for the OSGi environment so that components can be automatically published as services for the service registry, and references can be automatically resolved to services discovered from the service registry.

The Blueprint component model provides the simplicity of dependency injection frameworks such as the Spring Framework, including the ability to form a unit test that is separate from the server environment. Blueprint standardizes the configuration metadata, and brings governance to the specification of the component model.

The specification also describes how to use Java Naming and Directory Interface (JNDI) and Java Persistence API (JPA) in an OSGi framework, and web application bundles (WABs).

The WebSphere programming model and OSGi

The OSGi Applications programming model in WebSphere Application Server enables you to develop, assemble, and deploy modular applications that use Java EE and OSGi technologies. You can use tooling to deploy an enterprise application as an OSGi application that consists of one or more OSGi bundles.

The benefits of deploying an application as a set of bundles are described in “Business goals and OSGi Applications” on page 476.

The OSGi Applications programming model in WebSphere Application Server enables both new and existing applications to be deployed as OSGi applications. An existing enterprise application can be deployed as an OSGi application in which each component is converted to an OSGi bundle, as described in Converting an enterprise application to an OSGi application. You can develop new applications as OSGi Application projects as described in Developing an OSGi application. A new application might consist of two bundles, where each one contains business logic in plain old Java objects (POJOs), wired together through a Blueprint module definition, or by invoking an enterprise bean. The granularity of the bundle is such that each bundle has a coherent function in the context of an enterprise application. One bundle provides a service that the other bundle requires. The Blueprint Container in WebSphere Application Server wires the components in a bundle by creating component (bean) instances and injecting its dependencies. When a component in Bundle B, an EJB bundle, offers a service that Bundle A requires, the Blueprint Container takes care of registering the service in the service registry and injecting a service reference into the consuming component in Bundle A.

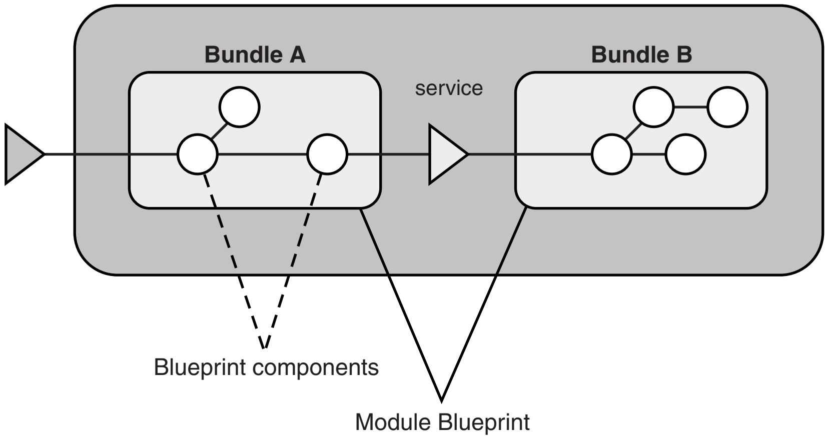


Figure 66. OSGi application with two bundles

For example, consider an application that represents a weblog service. A weblog business logic bundle exports a service that can be consumed by a web application bundle that handles HTTP clients for this service. The weblog bundle depends on two other bundles; one to provide persistence to a database, and one to provide a service so that readers of the weblog can post comments to the weblog. The weblog business logic bundle consists internally of three components whose configuration and references are injected into them at run time.

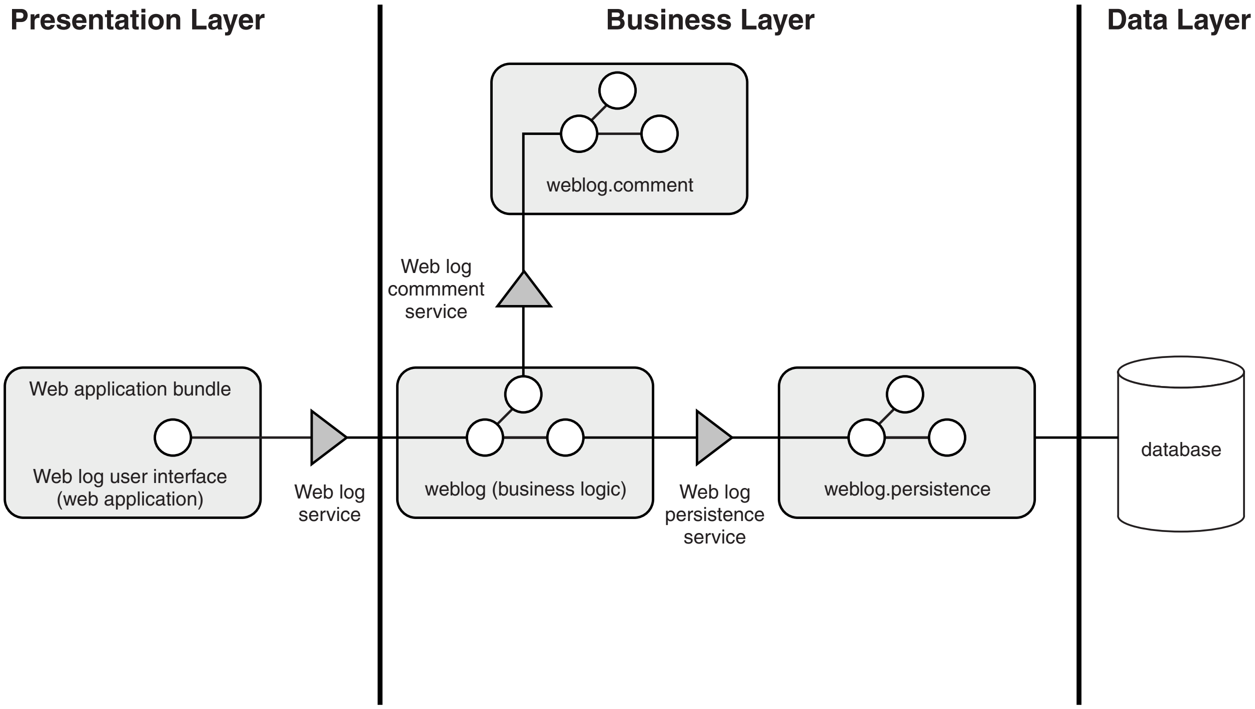


Figure 67. Example application for a weblog service

If you have used a dependency injection framework such as the Spring Framework, the Extensible Markup Language (XML) module Blueprint configuration for the components in the weblog business bundle will be familiar. Each component is defined, along with the references and configurations that need to be injected. Specifically:

- Each service that needs to be automatically published to the service registry is defined.
- Each reference that needs to be resolved automatically from the service registry for injection into a component is declared.

OSGi services provide a convenient way to represent dependencies between bundles. Services have the same life cycle as the bundle that provides them. The underlying server run time wires services dynamically when bundles are started and stopped.

Unit test

Unit test for Blueprint components is simplified by the dependency injection pattern, which allows one bean to access another bean without having to implement any code to create the bean instance. The Blueprint Container creates the required bean instance, using information contained in the Blueprint configuration file. This eliminates compiled dependencies on either the OSGi Framework or the application server runtime environment. In the weblog application example, you can write a Java unit test for the weblog component that can run in a simple Java SE environment, or an integrated development environment (IDE), with no need to install the component to its target runtime environment. The unit test can include the following actions:

- Creation of a new weblog comment bean, rather than creation by the Blueprint Container.
- Injection of the configuration that is relevant to the test.
- Testing the function of the weblog comment bean.

OSGi Applications support

An OSGi application is a collection of one or more OSGi modules that together provide a coherent business function. An OSGi application can consist of modules of many different types. For example, the weblog example described earlier might consist of bundles with web content (web application bundles), bundles with Blueprint contexts, EJB bundles, and bundles with JPA entities and persistence configuration (persistence bundles).

The modules that are contained in an OSGi application can offer OSGi services. The OSGi Applications support isolates those OSGi services so that they are not visible outside the application, unless they are explicitly configured to be exported from the application. OSGi applications have several ways to accept workloads:

- An OSGi application can include web bundles to process HTTP workloads.
- When the modules that are contained in an application offer OSGi services, an OSGi application can export one or more of those services to other OSGi applications, by using Service Component Architecture (SCA); for further information, see SCA and OSGi Applications.

An OSGi application isolates the OSGi services that are offered to modules that are contained in the application. The modules cannot consume services outside the application unless they are explicitly configured to import them. These imported services might be proxies to other OSGi application services or proxies to remote services (for example, web services); these proxies are provided by SCA.

In the following example, an OSGi application consists of three bundles. The application exposes one service, the web log service, and imports one service, the User authoring service. Within the application, the web log persistence service is exported by the weblog.persistence bundle and imported by the weblog bundle, but must not be exposed outside the application. The application isolation hides the web log persistence service from outside the application.

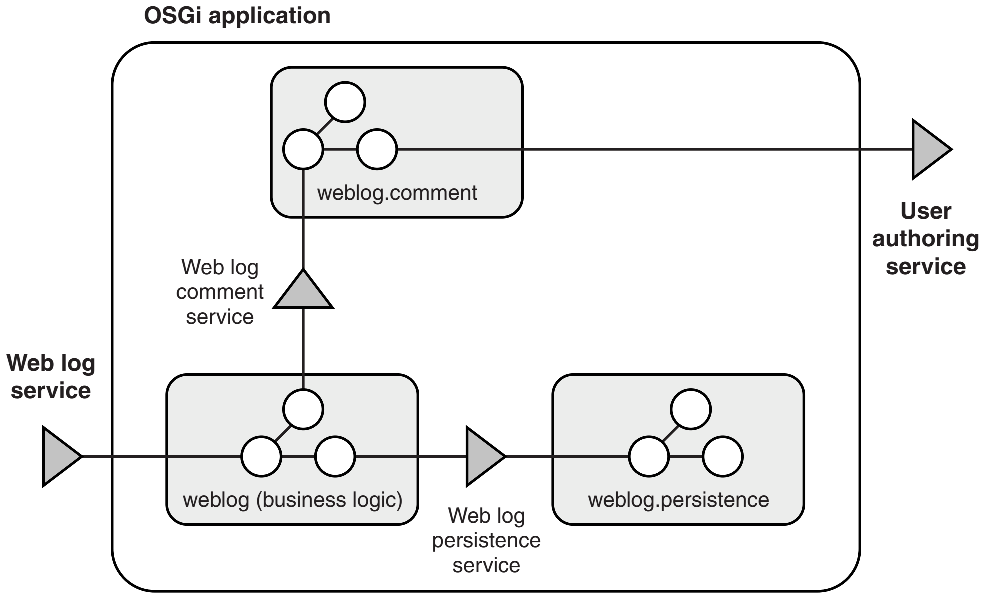


Figure 68. Bundles and services in an example application for a weblog service

OSGi Applications support provides a modularity construct at application level to describe an isolated application, including metadata that describes the constituent bundles of an application. A deployment system can use this to determine the constituent bundles that must be provisioned and deployed when the application is installed to a target server.

An OSGi application consists of a set of bundles, in an isolation scope that is defined by application metadata. An OSGi application is deployed as an enterprise bundle archive (EBA) file. This file contains the constituent bundles for the application, or the metadata required to get the constituent bundles from an OSGi bundle repository, or both.

If an OSGi application provides or requires any external services and references, these are explicitly made available by declaring them in an application manifest.

Any external services and references that the OSGi application must satisfy are also declared in the application manifest. The application manifest describes modularity at the application level in a similar way to OSGi headers in a bundle manifest file that define modularity at the bundle level.

The following examples illustrate the flexible ways in which an OSGi application can be deployed as an EBA file, with and without application-level isolation, and optionally exploiting a shared OSGi bundle repository. Each example extends the previous one.

- The OSGi application has an application-specific copy of each bundle on disk and in memory. Each bundle is at version 1.0.0.
- The OSGi application references one bundle from a bundle repository, requiring version 1.0.0 or later to be present in the bundle repository. There is one less bundle on disk. However, the same number of bundles are loaded into memory, because each bundle is isolated for each application in memory so there is still one copy of each bundle for the application in memory.
- The OSGi application contains all the application bundles, but a later version of one bundle is in the configured bundle repository and is within the version range in the application manifest. The installed application is provisioned to use the bundle from the bundle repository, rather than the bundle that is included in the application. An example of typical use is using a governed bundle repository to deliver critical fixes to shared bundles.
- A content bundle in the OSGi application uses a service provided by a bundle that is not part of the application content. Dependencies that are not inside the application are satisfied by bundles that are shared by all applications on the same server. This reduces the memory footprint for applications that do not require component instances from the dependency bundle to be isolated in the OSGi application. Equivalent sharing for Java EE and SCA is achieved by using an isolated shared library class loader with a JAR file, but this mechanism requires greater administrative overhead

The Blueprint Container

The Blueprint Container specification defines a dependency injection framework for OSGi and is an OSGi Alliance standard. It provides a simple programming model to create dynamic applications in the OSGi environment.

The Blueprint Container specification deals with the dynamic nature of OSGi, where services can become available and unavailable at any time. The specification also works with plain old Java objects (POJOs), so that the same objects can be used inside and outside the OSGi framework. For example, you can write and unit test simple components in a Java Platform, Standard Edition (JSE) environment without needing to know how they are assembled.

Key factors in the Blueprint programming model are the Blueprint XML files that define and describe the assembly of various components. The specification describes how the components are instantiated and wired together to form a module that runs.

This information describes frequently-used aspects of the Blueprint Container. For more detail, see the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification.

Blueprint bundles

A Blueprint bundle is a bundle that contains one or more Blueprint XML files. The Blueprint Container specification uses an extender pattern. An extender bundle monitors the state of bundles in the framework and acts on behalf of those bundles, based on the state of those bundles.

The Blueprint extender bundle waits for the bundles to be activated and checks whether each one is a Blueprint bundle. The Blueprint XML files are at a fixed location in the `OSGI-INF/blueprint/` directory, or are specified explicitly in the Bundle-Blueprint manifest header. When a bundle is a Blueprint bundle, the extender bundle creates a Blueprint Container for that bundle.

The Blueprint Container parses the Blueprint XML files, instantiates the components, wires components together, and registers services. During initialization, the Blueprint Container ensures that mandatory service references are satisfied, registers all the services into the service registry, and creates initial component instances.

When a Blueprint bundle is stopped, the Blueprint extender bundle destroys the Blueprint Container for that bundle.

Blueprint XML

A Blueprint XML file is identified by a top-level blueprint element and contains definitions of component managers such as a bean manager, a service manager, and service reference managers.

A Blueprint XML file is identified by a top-level Blueprint element, as shown in the following `blueprint.xml` example code.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
   ...
</blueprint>
```

The XML namespace identifies that the document conforms to the Blueprint version 1.0.0. The top-level Blueprint element identifies the document as a Blueprint module definition.

The Blueprint XML file contains definitions of various component managers. The Blueprint Container specification defines four main component managers:

- A bean manager creates an instance of a Java object with the given arguments and properties. A bean manager can create single or multiple object instances, depending on the scope settings. It can also manage the life cycle of an object and notify it when all properties have been injected or when it is being destroyed.
- A service manager registers and unregisters a service in the OSGi service registry.
- Two service reference managers provide access to the services registered in the OSGi service registry:
  - A reference manager provides an object that is a proxy to the service that is registered in the service registry.
  - A reference-list manager provides a dynamic list of either service proxy objects or service reference objects that are currently in the service registry.

Each component manager creates components and manages the life cycle of those components. When requested, the managers provide a component instance. Each manager has a corresponding XML element that describes the manager properties. The managers can be top-level managers or they can be defined by using inline declarations in other manager definitions. All component managers can have the following attributes:

id This optional attribute defines the ID of a top-level manager. The ID must be unique for all top-level managers in the Blueprint Container. If you do not specify this attribute, a unique ID is generated automatically. Managers use the ID to refer to each other. For example, during injection, the manager asks the referenced managers to provide an object that is injected into the component that the manager is creating.

You do not set the id attribute for managers that are defined by using inline declarations in other manager definitions, because these managers are considered to be anonymous.

activation

This optional attribute defines the activation mode of the manager. The following activation modes are supported:

- eager. The manager is activated during initialization of the Blueprint Container. This is the default.

  A service manager is published into the service registry and activates the bean manager that underlies the service.
- lazy. The manager is activated on demand. A manager is activated when it is requested to provide its first component instance.

  For a bean manager, the bean is instantiated only when another bean manager first accesses it. For a service manager, the service manager is published into the service registry, but it does not activate the bean manager that underlies the service.

To change the default activation mode for all managers in the Blueprint XML file, set the default-activation attribute on the Blueprint element.

Each manager has its own activation and deactivation steps. A manager is deactivated when the Blueprint Container is destroyed.

dependsOn

This optional attribute defines the explicit dependencies of a manager. It specifies a list of manager IDs, where those managers must be activated before this manager is activated. A manager can also have implicit dependencies, which are defined by the references to other managers in a manager definition.

Beans and the Blueprint Container

In the Blueprint programming model, you declare beans by using the bean element. You specify argument elements to provide the arguments that are used for object construction, and you specify property elements to provide the injected properties.

You can specify the value of argument and property elements by using a value or ref attribute, or you can use an inline declaration in an element. The ref attribute specifies the ID of a top-level manager, and is used to obtain an object from the referenced manager as the argument or property value. The inline value can be any XML value that is described in “Object values and the Blueprint Container” on page 492.

The following `bean.xml` example code defines a single bean called accountOne that is implemented by the org.apache.aries.simple.Account plain old Java object (POJO).

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
   <bean id="accountOne" class="org.apache.aries.simple.Account" />
</blueprint>
```

Bean Construction

To construct an object, first the Blueprint Container finds the correct constructor or factory method with a compatible set of parameters that match the arguments that are specified in the XML. By default, the Blueprint Container uses the number and order of the argument elements in XML to find the correct constructor or method. If the argument elements in their current order do not map to the parameters, the Blueprint Container reorders the argument elements and attempts to find the best-fitting arrangement.

You can specify additional attributes, such as index or type, on the argument element so that it is easier for the Blueprint Container to find the correct constructor, method, or parameter arrangement. For example, the type attribute specifies a class name used to match the argument element to a parameter by the exact type.

You can construct a bean in either of the following ways:

- Use a class constructor.
- Use a static factory method .
- Use an instance factory method.

The following partial Java class and Blueprint XML example code shows how to construct a bean by using a class constructor. The class attribute specifies the name of the Java class to instantiate. The Blueprint Container creates the Account object by passing the value 1 as the argument to the constructor.

```
public class Account {
   public Account(long number) {
      ...
   }
   ...
}
```

```
<bean id="accountOne" class="org.apache.aries.simple.Account">
   <argument value="1"/>
</bean>
```

The following partial Java class and Blueprint XML example code shows how to construct a bean by using a static factory method. The class attribute specifies the name of the class that contains a static factory method. The factory-method attribute specifies the name of the static factory method. The Blueprint Container calls the createAccount() static method on the StaticAccountFactory class and passes the value 2 as the argument to create the Account object.

```
public class StaticAccountFactory {
   public static Account createAccount(long number) {
      return new Account(number);
   }
}
```

```
<bean id="accountTwo" class="org.apache.aries.simple.StaticAccountFactory"
   factory-method="createAccount">
   <argument value="2"/>
</bean>
```

The following partial Java class and Blueprint XML example code shows how to construct a bean by using an instance factory method. You use two managers; one manager is a factory, and the other uses the factory to create an object. The factory-ref attribute specifies the ID of a top-level bean or a reference manager that acts like a factory. The provided factory object must have a factory method, as specified by the factory-method attribute.

The accountFactory bean is the factory. The Blueprint Container first creates the AccountFactory instance with its own arguments and properties. In this example, a single argument, the factory name, is specified. The Blueprint Container then calls the createAccount() method on the AccountFactory instance and passes the value 3 as the argument to create the Account object.

```
public class AccountFactory {
   public AccountFactory(String factoryName) {
      ...
   }
   public Account createAccount(long number) {
      return new Account(number);
   }
}
```

```
<bean id="accountFactory" class="org.apache.aries.simple.AccountFactory">
   <argument value="account factory"/>
</bean>
```

```
<bean id="accountThree"
      factory-ref="accountFactory"
      factory-method="createAccount">
   <argument value="3"/>
</bean>
```

Bean properties

You can use the property element to inject property values into beans. Properties are injected immediately after the bean is constructed. The following partial Java class and Blueprint XML example code creates the Account bean, then sets the description property by using the Java Beans naming convention.

```
public class Account {
   public Account(long number) {
      ...
   }
   public void setDescription(String desc) {
      ...
   }
}
```

```
<bean id="accountOne" class="org.apache.aries.simple.Account">
   <argument value="1"/>
   <property name="description" value="#1 account"/>
</bean>
```

You can use property injection to wire beans together. In the following Blueprint XML example code, the accountOne bean is injected with a Currency bean.

```
public class Account {
   public Account() {
      ...
   }
   public void setCurrency(Currency c) {
      ...
   }
}
```

```
public class Currency {
   public Currency() {
      ...
   }
}
```

```
<bean id="accountOne" class="org.apache.aries.simple.Account">
   <property name="currency" ref="currency" />
</bean>

<bean id="currency" class="org.apache.aries.simple.Currency" />
```

Configuring bean security

Optionally, you can configure bean security so that the methods of the bean can be accessed only by users that are in a specified role.

You configure security by specifying an <access-constraint> element. The <access-constraint> element has a role attribute that defines the name of the role.

To define access controls for a particular role, use the Security role to user or group mapping panel in the WebSphere Application Server administrative console.

In the following example, the methods of the secureBean1 bean are accessible only by users in the role called "ROLE1":

```
<bean
  id="secureBean1"
  class="com.ibm.ws.eba.wab.componenttest.blueprint.secure.BlueprintSecureServiceImpl">
  <access-constraint role="ROLE1" />
</bean>
```

For the bean security configuration to be effective, application security must be enabled in WebSphere Application Server.

If you do not specify an <access-constraint> element, the bean is not secured; this means that all the methods of the bean can be accessed by any user.

Services and the Blueprint Container

In the Blueprint programming model, you use a service element to define the registration of a service in the OSGi service registry. You use the ref attribute to reference the bean that provides the service object. You use the interface attribute to specify the interfaces under which the service is registered.

Note: The service registration mechanism described in this topic is not required if you want to define an EJB as a service. Instead, specify the EJB class name in the Export-EJB header in the bundle manifest file.

See the following partial Java class and Blueprint XML example code.

```
public class AccountImpl implements Account {
   public AccountImpl() {
      ...
   }
}
```

```
<service id="serviceOne" ref="account"
   interface="org.apache.aries.simple.Account" />

<bean id="account" class="org.apache.aries.simple.AccountImpl" />
```

You can specify the bean that provides the service object by using an inline declaration in the service element, as shown in the following Blueprint XML example code.

```
<service id="serviceTwo" interface="org.apache.aries.simple.Account">
   <bean class="org.apache.aries.simple.AccountImpl" />
</service>
```

You can use the auto-export attribute to set the interfaces under which a service is registered. The following Blueprint XML example code registers the service under all the interfaces of the bean.

```
<service id="serviceOne" ref="account" auto-export="interfaces" />

<bean id="account" class="org.apache.aries.simple.AccountImpl" />
```

The default value for the auto-export attribute is disabled. Other values are class-hierarchy and all-classes.

Service properties

You can register a service with a set of properties by using the service-properties element. The service-properties element contains multiple entry elements that represent the individual properties. You specify the property key by using a key attribute. You specify the property value as a value attribute, or in inline declaration in the element. Service property values can be different types, but must be only OSGi service property types, that is, one of the following types:

- primitive
- primitive wrapper class
- collection
- array of primitive types

The following Blueprint XML example code shows a service registration with two service properties. The active service property has type of java.lang.Boolean. The mode property is of the default type, String.

```
<service id="serviceFour" ref="account" autoExport="all-classes">
   <service-properties>
      <entry key="active">
         <value type="java.lang.Boolean">true</value>
      </entry>
      <entry key="mode" value="shared"/>
   </service-properties>
</service>
```

Service ranking

You can use service ranking to control the choice of service when there are multiple matches. If there are two services, the higher ranked service is returned before the lower ranked one. The default ranking value is 0. The following Blueprint XML example code shows how to specify service ranking by using the ranking attribute.

```
<service id="serviceFive" ref="account" auto-export="all-classes" ranking="3" />
```

References and the Blueprint Container

In the Blueprint programming model, you use the reference element to find services in the service registry, and the reference-list element to find multiple matching services.

The reference element is used to refer to a service in the OSGi service registry, and is independent of how that service was registered. For example, it can refer to the following services:

- A Java bean that is declared in a service element in the Blueprint XML file of the bundle that provides the service; for more information, see “Services and the Blueprint Container” on page 488.
- An enterprise bean that is specified in the Export-EJB header in the manifest file of the bundle that provides the service.
- A service that is registered by making direct calls to the OSGi API.

The following Blueprint XML example code shows the accountRef reference that refers to an Account service. If a service that matches this reference is found in the service registry, the service is set on the accountClient bean through the account property.

```
<bean id="accountClient" class="...">
   <property name="account" ref="accountRef" />
</bean>

<reference id="accountRef" interface="org.apache.aries.simple.Account" />
```

Reference dynamism

The object that is injected for a reference is a proxy to the service that is registered in the service registry. By using a proxy, the injected object remains the same when the availability of the backing service varies, or the backing service is replaced. If there are calls on a proxy that does not have a backing service, those calls block until a service becomes available or a timeout occurs. The default timeout in the Blueprint component is 300000 milliseconds (5 minutes). If a timeout occurs, a ServiceUnavailableException exception is created and the Blueprint Container is destroyed. See the following example code.

```
try {
   balance = account.getBalance();
} catch (ServiceUnavailableException e) {
   ...
}
```

You can change the timeout for each bundle by using directives in the Bundle-SymbolicName header in the bundle manifest, `META-INF/MANIFEST.MF`. The following example switches the timeout for a bundle off (the default is true).

```
Bundle-SymbolicName: org.apache.aries.simple.account;
   blueprint.graceperiod:=false
```

The following example sets the timeout for a bundle to 10000 milliseconds (10 seconds).

```
Bundle-SymbolicName: org.apache.aries.simple.account; blueprint.timeout=10000;
```

You can set the timeout for an individual reference by using the timeout attribute. The following example XML code sets the timeout for the accountRef reference to 20000 milliseconds (20 seconds).

```
<reference id="accountRef" timeout="20000"
   interface="org.apache.aries.simple.Account" />
```

For both a bundle and a reference, a timeout value of 0 means wait indefinitely for the reference to be satisfied.

Reference lists

You can use the reference-list element to find multiple matching services. The reference-list element provides a List object that contains the service proxy objects or ServiceReference objects, depending on the value of the member-type attribute (the default value is service-object). The List object that is provided is dynamic and expands or contracts when matching services are added or removed from the service registry. The List object is read-only and supports only a subset of the List API.

The proxies in a reference-list element target a specific service and do not have a timeout. If the backing service for a proxy becomes unavailable, a ServiceUnavailableException exception is created immediately.

The following Blueprint XML example code shows a reference-list that returns a list of service objects (proxies).

```
<reference-list id="accountRefs" member-type="service-object"
   interface="org.apache.aries.simple.Account" />
```

The following Blueprint XML example code shows a reference-list that returns a list of ServiceReference objects.

```
<reference-list id="accountRefs" member-type="service-reference"
   interface="org.apache.aries.simple.Account" />
```

Availability

By default, a service reference manager requires that at least one suitable service exists before initialization of the Blueprint Container can continue. That is, a service that matches the selection criteria of the reference or reference list must exist. To control this behavior, you use the availability attribute of the reference or reference-list element. The availability attribute can have the following values:

optional
The existence of a service that matches the selection criteria of the element during initialization of the Blueprint Container is optional. A service reference manager with optional availability is always considered satisfied, even if it does not have any matching services.

mandatory
At least one service that matches the selection criteria of the element during initialization of the Blueprint Container must exist. For a service reference manager with mandatory availability, if it has a matching service, it is considered satisfied. This is the default.

Initialization of the Blueprint Container is delayed until all service reference managers with mandatory availability are satisfied.

The availability attribute applies only during initialization of the Blueprint Container. After initialization, a service reference manager that has mandatory availability can become unsatisfied at any time, when services become unavailable or available again.

To change the default availability setting for all service reference managers in the Blueprint XML, you use the default-availability attribute on the Blueprint element.

The following Blueprint XML example code shows a reference manager that has mandatory availability.

```
<reference id="accountRef" timeout="20000"
   interface="org.apache.aries.simple.Account"
   availability="mandatory"/>
```

Service selection and proxies

The reference and reference-list managers share the following attributes that you can use to select services:

interface
Use the interface attribute to specify an interface class. The interface attribute is optional, but if set, it must specify an interface class. The interface class is used to select services or return service proxies.

For service selection, the interface class is used to select services from the service registry registered with that interface name. For service proxies, the proxies that the service reference managers return must implement all the methods that the interface class defines. If the interface attribute is not specified, the proxy behaves as though it implements an interface without any methods.

component-name
Use the component-name attribute to select services by adding an `osgi.service.blueprint.compname=`*`component_name`* expression to the selection filter.

filter
Use the filter attribute to select services by specifying the raw OSGi filter expression to add to the selection filter.

The three attributes combine to create one OSGi filter expression to use to select services.

For example, the selection filter for the reference in the following Blueprint XML example code is `(&(objectClass=org.apache.aries.simple.Account) (osgi.service.blueprint.compname=myAccount)(mode=shared))`.

```
<reference id="accountRef"
   interface="org.apache.aries.simple.Account"
   component-name="myAccount"
   filter="(mode=shared)"/>
```

Scopes and the Blueprint Container

In the Blueprint programming model, you use the scope setting to determine whether a bean manager creates single or multiple object instances.

The Blueprint Container specification defines two scopes:

singleton
The bean manager creates a single instance of the bean and returns that instance every time that the manager is requested to provide an object. This is the default for top-level bean managers.

prototype
The bean manager creates a new instance of the bean every time that the manager is requested to provide an object. This is the default for bean managers that are specified in an inline declaration.

The following Blueprint XML example code shows how to set a singleton scope for a bean manager.

```
<bean id="singletonAccount" class="org.apache.aries.simple.Account"
   scope="singleton">
   <argument value="5"/>
</bean>
```

The following Blueprint XML example code shows how to set a prototype scope for a bean manager.

```
<bean id="prototypeAccount" class="org.apache.aries.simple.Account"
   scope="prototype">
   <argument value="4"/>
</bean>
```

Object values and the Blueprint Container

The Blueprint Container specification defines XML elements that describe different types of object values and that you can use in manager definitions.

For example, you can use XML value elements in a bean manager to specify argument or property values, or in a service manager to specify the values of service properties. The XML value elements are converted into value objects and injected into the manager components. You can use the following XML value elements:

ref The ref element defines a reference to a top-level manager. The component-id attribute specifies an ID of a top-level manager. The injected value will be the object that the referenced manager returns.

The following partial Java class and Blueprint XML example code shows an example of the ref value element. The accountOne bean instance is injected into the managedAccount property of the accountManagerTwo bean.

```
public class AccountManager {
   ...
   public void setManagedAccount(Account account) {
      ...
   }
}
```

```
<bean id="accountOne" class="org.apache.aries.simple.Account">
   <argument value="1"/>
   <property name="description" value="#1 account"/>
</bean>

<bean id="accountManagerTwo" class="org.apache.aries.AccountManager">
   <property name="managedAccount">
       <ref component-id="accountOne"/>
   </property>
</bean>
```

idref The idref element defines an ID of a top-level manager. The injected value is the component-id, as specified by the component-id attribute. The idref element is used to ensure that a manager with the specified ID actually exists before the manager is activated.

value The value element represents an object to create from the string content of the element. Optionally, use the type attribute to specify a type to convert the string content to. If you do not specify the type attribute, the string content is converted to the type that it is injected into.

null The null element represents Java null.

list The list element is a collection and represents a java.util.List object. Any XML value element that is described in this section can be a sub-element of this collection. Optionally, you can set the value-type attribute to specify a default type for the collection sub-elements.

set The set element is a collection and represents a java.util.Set object. Any XML value element that is described in this section can be a sub-element of this collection. Optionally, you can set the value-type attribute to specify a default type for the collection sub-elements.

array The array element is a collection and represents an Object[] array. Any XML value element that is described in this section can be a sub-element of this collection. Optionally, you can set the value-type attribute to specify a default type for the collection sub-elements.

The following Blueprint XML example code shows how you can combine XML value elements to create a list. The created list will contain the following items:

- The string `123`
- A java.math.BigInteger object with a value of `456`
- A null
- A java.util.Set object with two values that are of type java.lang.Integer

```
<list>
   <value>123</value>
   <value type="java.math.BigInteger">456</value>
   <null/>
   <set value-type="java.lang.Integer">
      <value>1</value>
      <value>2</value>
   </set>
</list>
```

props The props element represents a java.util.Properties object where the keys and values are of String type. The prop sub-elements represent the individual properties. To specify the property key, you use a key attribute, and to specify the property value, you use a value attribute or specify the content of the element.

The following Blueprint XML example code shows an example of the props value element.

```
<props>
   <prop key="yes">good</prop>
   <prop key="no" value="bad"/>
</props>
```

map The map element represents a java.util.Map object where the keys and the values can be arbitrary objects. The entry sub-elements represent the individual properties.

To specify the key, you use a key or key-ref attribute, or an inline declaration in a key sub-element. You can specify an inline key by using any XML value element that is described in this section, except the null element. In the Blueprint Container specification, map elements cannot have null keys.

To specify the value, you use a value or value-ref attribute, or an inline declaration. You can specify an inline value using any XML value element that is described in this section. However, the inline key can be specified as any of the XML value elements except the null element.

The key-ref and value-ref attributes specify an ID of a top-level manager and are used to obtain an object from the specified manager as the property key or value. The map element can specify key-type and value-type attributes to define a default type for keys and values.

The following Blueprint XML example code shows an example of the map value element and how the entries of a map object can be constructed in several ways. The created map object will contain the following entries:

- A myKey1 String key that is mapped to myValue String.
- A key that is an object that the account bean manager returns and that is mapped to myValue String.
- A key that is a java.lang.Integer object with a value of 123 and that is mapped to myValue String.
- A myKey2 String key is mapped to a value that is an object that the account bean manager returns.
- A myKey3 String key is mapped to a value that is a java.lang.Long object with a value of 345.

- A key that is a java.net.URI object with a value of urn:ibm and that is mapped to a value that is a java.net.URL object with a value of http://ibm.com value.

```
<map>
   <entry key="myKey1" value="myValue"/>

   <entry key-ref="account" value="myValue"/>
   <entry value="myValue">
      <key>
         <value type="java.lang.Integer">123</value>
      <key/>
   </entry>

   <entry key="myKey2" value-ref="account">
   <entry key="myKey3">
      <value type="java.lang.Long">345</value>
   </entry>

   <entry>
      <key>
         <value type="java.net.URI">urn:ibm</value>
      <key/>
      <value type="java.net.URL">http://ibm.com</value>
   </entry>
</map>
```

Each manager can also be specified as a value by using an inline declaration. The following Blueprint XML example code shows an inline bean manager.

```
<bean id="accountManagerThree" class="org.apache.aries.AccountManager">
   <property name="managedAccount">
      <bean class="org.apache.aries.simple.Account">
         <argument value="10"/>
         <property name="description" value="Inline Account"/>
      </bean>
   </property>
</bean>
```

Object life cycles and the Blueprint Container

In the Blueprint programming model, a bean manager can manage the life cycle of the object that it creates. The bean manager can notify an object after all properties are injected, or when an object instance is destroyed.

The Blueprint Container specification defines the following callback methods:

init-method
Specifies a method to be call after all properties are injected into an object. The method must be public, does not take any arguments, and does not return any value.

destroy-method
Specifies a method to call when the Blueprint Container destroys the object instance. The method must be public, does not take any arguments, and does not return any value.

The destroy-method callback is not supported for beans with a scope of prototype. In this situation, the application is responsible for destroying those instances.

The following code examples show an example of a Java class with lifecycle methods and a Blueprint XML bean entry that specifies the init-method and destroy-method attributes.

```
public class Account {
   public Account(long number) {
      ...
   }
   public void init() {
        ...
    }
    public void destroy() {
        ...
    }
}
<bean id="accountFour" class="org.apache.aries.simple.Account"
    init-method="init" destroy-method="destroy">
    <argument value="6"/>
    <property name="description" value="#6 account"/>
</bean>
```

Resource references and the Blueprint Container

Blueprint components can access WebSphere Application Server resource references by using resource reference declarations.

You declare each resource reference in a Blueprint XML file. Each bundle in an OSGi application can contain any number of resource reference declarations in its various Blueprint XML files. You can secure each resource reference by using a Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) authentication alias.

Blueprint resource reference bindings are configured when you add the enterprise bundle archive (EBA) asset to a business-level application.

To use resource references, you use the following Blueprint namespace, which contains the resource-reference, res-auth, and res-sharing-scope elements:

```
http://www.ibm.com/appserver/schemas/8.0/blueprint/resourcereference
```

The following example defines a Blueprint component with a component ID of resourceRef. The component is like a service reference. This component will be bound to a Java Naming and Directory Interface (JNDI) resource that has the JNDI name jdbc/Account and type javax.resource.cci.ConnectionFactory.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" xmlns:rr=
    "http://www.ibm.com/appserver/schemas/8.0/blueprint/resourcereference">
    <rr:resource-reference id="resourceRef"
        interface="javax.resource.cci.ConnectionFactory"
        filter="(osgi.jndi.service.name=jdbc/Account)">
        <rr:res-auth>Application</rr:res-auth>
        <rr:res-sharing-scope>Shareable</rr:res-sharing-scope>
    </rr:resource-reference>
</blueprint>
```

To declare a resource reference, you must specify both the interface and filter attributes. The filter must contain a JNDI resource in the form osgi.jndi.service.name=*name*, where *name* is the JNDI name of the resource.

Use the res-auth element to set whether authorization is application-managed or container-managed. The res-auth element takes the following values:

Application
Authorization is application-managed.

Container
Authorization is container-managed. This is the default value.

Use the res-sharing-scope element to set whether a resource reference can be shared. The res-sharing-scope element takes the following values:

Shareable
A resource reference can be shared

Unshareable
A resource reference cannot be shared. This is the default value.

You can also use a resource reference to configure a data source for a persistence bundle by using a blueprint:comp/*resource_ref_id* JNDI name. For more information, see "JPA and OSGi Applications" on page 520.

When you secure resource references, those resource references can be bound only to JCA authentication aliases that exist on every server or cluster that the OSGi application is deployed to. An OSGi application can be deployed to multiple servers and clusters that are in the same security domain. Therefore, each JCA authentication alias must exist in either the security domain of the target servers and clusters, or the global security domain.

Dynamism and the Blueprint Container

In the Blueprint programming model, you can use registration listeners or reference listeners so that a bundle can have control when services become available.

For example, a bundle might have an optional service that it wants to start using when the service becomes available, and stop using when the service becomes unavailable. You can use a reference listener so that the bundle is notified when this optional service becomes available or unavailable.

The service reference managers, that is, a reference manager or a reference-list manager, can have zero or more registration listeners, and zero or more reference listeners.

- Registration listeners are objects that have callback methods invoked immediately after a service is registered, or immediately before a service is unregistered.
- Reference listeners are objects that have callback methods invoked when a service is selected by the service reference manager, or a service is no longer used by the service reference manager.

Registration listener

To specify a registration listener, use the registration-listener element, and use the registration-method and unregistration-method attributes specify the callback methods. You can specify the object that provides the callback methods as a reference to a top-level manager or you can use an inline declaration in the registration-listener element.

If the service implements the ServiceFactory interface, the registration and unregistration callback methods must have the following signature, where *anyMethod* represents an arbitrary method name.

```
void anyMethod(ServiceFactory, Map)
```

If the service does not implement the ServiceFactory interface, the registration and unregistration callback methods must match the following signature.

```
void anyMethod(? super T, Map)
```

The first argument is an instance of the service object. The type T must be able to be assigned from the type of the service object. The second argument provides the registration properties that are associated with the service.

If a registration listener has multiple overloaded methods for a callback, every method with a matching signature is invoked.

The following partial Java class and Blueprint XML example code shows a simple registration listener.

```
public class RegistrationListener {
   public void register(Account account, Map properties) {
      ...
   }
   public void unregister(Account account, Map properties) {
      ...
   }
}
```

```
<service id="serviceSix" ref="myAccount" auto-export="all-classes">
   <registration-listener
      registration-method="register" unregistration-method="unregister">
      <bean class="org.apache.aries.RegistrationListener"/>
   </registration-listener>
</service>
```

Reference listener

To specify a reference listener, you use the reference-listener element, and use the bind-method and unbind-method attributes to specify the callback methods. You can specify the object that provides the callback methods as a reference to a top-level manager or you can use an inline declaration in the reference-listener element.

The bind and unbind callback methods can have any of the following signatures, where *anyMethod* represents an arbitrary method name:

- `void anyMethod(ServiceReference)`

  The argument is a ServiceReference object of the service that is bound or unbound.
- `void anyMethod(? super T)`

  The argument is the service object proxy that is bound or unbound. The type T must be able to be assigned from the service object.
- `void anyMethod(? super T, Map)`

  The first argument is the service object proxy that is bound or unbound. The type T must be able to be assigned from the service object. The second argument provides the service properties that are associated with the service.

If a reference listener has multiple overloaded methods for a callback, every method with a matching signature is invoked.

For a reference-list manager, the listener callbacks are invoked each time a matching service is added or removed from the service registry. However, for a reference manager, the bind callback is not invoked when the manager is already bound to a service and a matching service with lower ranking is added to the service registry. Similarly, the unbind callback is not called if the service that the manager is bound to goes away and it is replaced immediately with another matching service.

If you use a reference manager and interact with stateful services, it is important to use reference listeners to track the backing services of the proxy so that you can manage the state of the service appropriately.

The following partial Java class and Blueprint XML example code shows a simple registration listener. The ReferenceListener class has two bind and one unbind callback methods, which are called when the services are bound and unbound from the service reference-list manager.

```
public class ReferenceListener {
   public void bind(ServiceReference reference) {
      ...
   }
   public void bind(Serializable service) {
      ...
   }
   public void unbind(ServiceReference reference) {
      ...
   }
}
```

```
<reference-list id="serviceReferenceListTwo"
   interface="org.apache.aries.simple.Account" availability="optional">
   <reference-listener bind-method="bind" unbind-method="unbind">
      <bean class="org.apache.aries.ReferenceListener"/>
   </reference-listener>
</reference-list>
```

Type converters and the Blueprint Container

During injection, the Blueprint Container converts the XML value elements into value objects that are injected into the manager components. The elements are converted based on the type of the injected property.

The Blueprint Container provides the following built-in conversions:

-
  - String values can be converted to all primitive types, wrapper types, or any types that have a public constructor that takes a String value.
  - Array elements can be converted to collection objects with compatible member types.
  - List or set elements can be converted to array objects with compatible member types.

The Blueprint Container also supports generics. If the generics information is available, the Blueprint Container uses that information for the conversions. For example, in the following Blueprint XML example code, the list element is converted into a list of java.util.Long objects.

```
public class AccountManager {
   ...
      public void setAccountNumbers(List<Long> accounts) {
         ...
      }
}
```

```
<bean id="accountManagerFour" class="org.apache.aries.AccountManager">
   <property name="accountNumbers">
      <list>
         <value>123</value>
         <value>456</value>
         <value>789</value>
      </list>
   </property>
</bean>
```

A Blueprint bundle can also provide its own converters. The custom converters are bean managers that provide an object that implements the Blueprint Converter interface. Specify the custom converters in the type-converters element under the Blueprint element. When the Blueprint Container is initialized, the type converters are initialized first, so that other managers can use the custom converters. For further details, see the Blueprint Container specification.

JNDI lookup for blueprint components

If a bundle contains blueprint XML that declares a number of components each with a given ID, those components can be looked up using the Java Naming and Directory Interface (JNDI).

The code that you use has the following form:

```
Object component = new InitialContext().lookup("blueprint:comp/componentId");
```

This mechanism is useful in two different contexts:

- You can declare and configure any number of components of a web application bundle (WAB), which can then be looked up from servlets that are not themselves blueprint-managed.
- You can declare and configure data sources in blueprint XML, then reference them in a `persistence.xml` file.

Declaring and configuring components of a WAB

A WAB can contain blueprint XML. This can be used to declare and configure any number of components, which can then be looked up from servlets that are not themselves blueprint-managed. One particular use of this is shown in the OSGi blog sample application, in which the web bundle contains the following blueprint code:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <reference id="blogService"
             interface="com.ibm.samples.websphere.osgi.blog.api.BloggingService"/>
</blueprint>
```

The JNDIHelper.getBloggingService() method call includes this code fragment:

```
  try {
    InitialContext ic = new InitialContext();
    return (BloggingService) ic.lookup("blueprint:comp/blogService");
  } catch (NamingException e) {
```

This code looks up a blueprint-managed reference to an OSGi service. This is useful because blueprint-managed services are *damped* as described in section 121.10.1 of the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification. If the BloggingService object is not available when the application code tries to use the reference, the application code waits until the service becomes available again.

Previous versions of the getBloggingService method used a lookup of the following form:

```
ic.lookup("osgi:service/com.ibm.samples.websphere.osgi.blog.api.BloggingService");
```

This is not so useful, because the application code can receive a `ServiceUnavailableException` exception if the service is not available when the application code tries to use the BloggingService object. For example: If the BloggingService object is temporarily unavailable because an in-place update is in progress, the user of the Blog web application can get an `HTTP 500 (Internal Error)` message in their web browser. With the new form of lookup, web requests wait for a short time (a second or two) until the update completes. Therefore it is easier to write a web application that remains continuously available, from a user perspective, even while the application is being updated in place.

Declaring and configuring data sources

Data sources can be declared and configured in blueprint XML, then referenced in a `persistence.xml` file. For example, in Apache Aries the `org.apache.aries.jpa.container.itest.bundle/src/main/resources/OSGI-INF/blueprint/config.xml` file includes the following code:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="nonjta" class="org.apache.derby.jdbc.EmbeddedDataSource">
    <property name="databaseName" value="memory:testDB"/>
    <property name="createDatabase" value="create"/>
  </bean>

  <service interface="javax.sql.XADataSource">
    <service-properties>
      <entry key="transactional" value="true"/>
    </service-properties>
    <bean class="org.apache.derby.jdbc.EmbeddedXADataSource">
      <property name="databaseName" value="memory:testDB"/>
      <property name="createDatabase" value="create"/>
    </bean>
  </service>

  <reference id="jta" availability="optional" interface="javax.sql.DataSource"
             filter="(transactional=true)"/>

</blueprint>
```

The `org.apache.aries.jpa.container.itest.bundle/src/main/resources/META-INF/persistence.xml` file includes the following corresponding code:

```
...
  <persistence-unit name="bp-test-unit" transaction-type="JTA">
    <description>Test persistence unit for the JPA Container and Context iTests</description>
    <jta-data-source>blueprint:comp/jta</jta-data-source>
    <non-jta-data-source>blueprint:comp/nonjta</non-jta-data-source>
    <class>org.apache.aries.jpa.container.itest.entities.Car</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <!-- These properties are creating the database on the fly. -->
      <!-- We are using them to avoid the tests having to create a database  -->
      <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema(ForeignKeys=true)"/>
      <property name="openjpa.jdbc.DBDictionary" value="derby"/>
    </properties>
  </persistence-unit>
</persistence>
```

In this code fragment, the `jta-data-source` and `non-jta-datasource` elements are configured through `blueprint:comp/` namespace references.

OSGi bundles and bundle archives

Applications can be deployed as versioned OSGi bundles, with each bundle typically having the granularity of an enterprise application module (for example, a webapplication). A bundle is a Java archive (JAR) or web application archive (WAR) file with standard OSGi metadata that describes aspects of the bundle, including the Java packages that the bundle exports, the Java packages that the bundle requires, and the bundle version.

Enterprise bundle archives

An enterprise bundle archive (EBA) file contains a set of OSGi bundles that are deployed as a single OSGi application, and that are isolated from other OSGi applications.

Each OSGi application runs in its own isolated OSGi framework instance with its own OSGi service registry. Bundles in one OSGi application cannot see bundles, services, or packages that are defined in another OSGi application, unless the bundles, services, or packages are explicitly shared by both applications.

The bundles that are used by the OSGi application are either directly contained in the EBA file, or pulled in by reference from an OSGi bundle repository when the application is provisioned. Application metadata stored in the EBA file defines the isolation scope of the bundles that the OSGi application uses.

An EBA file is packaged as a single compressed archive file with a `.eba` file extension. It can contain an application manifest, and a set of application modules. An application module can be one of the following types of resource:

- An OSGi bundle, packaged as a Java archive (JAR) file.

- A composite bundle, packaged as a composite bundle archive (CBA) file.
- A web application, packaged as a web application archive (WAR) file.

Application modules are contained in the root of the EBA file. These modules are directly contained in the OSGi application, whereas all other bundles are provided by reference.

An OSGi application can also use metadata to permit some of its constituent bundles to be shared. Sharing in this way can reduce the memory and resource requirements of a system. Shared bundles must be provided by reference rather than contained directly in an application.

An OSGi application can also load packages and consume OSGi services from a shared bundle space, that is, from the OSGi framework instance that is the parent of all the isolated framework instances of the OSGi applications.

Application manifest

The OSGi bundles in an EBA file share services with other OSGi applications by declaring them in an application manifest file, `META-INF/APPLICATION.MF`. Any external services and references that the OSGi application produces are exposed by declaring them in the manifest, and any external services and references that the application consumes are also declared in the manifest.

The application manifest specifies the bundles that form the primary content of the application. A bundle that is listed in the primary content might use a package that is not included in the application, and therefore require other bundles to be pulled in. The application manifest might also specify an allowed version range for some bundles; this range defines the initial version of the application, and the latest version to which it can be upgraded.

Optionally, you can explicitly declare shared bundles on which the primary application content depends so that you can restrict the level at which sharing is possible; for example, you can ensure that the application uses a shared bundle only between a specified version range. For more information, see the description of the "Use-Bundle" header in “Example: OSGi application manifest file” on page 512.

An EBA file does not have to contain an application manifest file. The contents of the EBA file is used in two different ways, depending on whether you have included an application manifest, and on whether an Application-Content header is defined in the application manifest:

- If an Application-Content header is not defined, or there is no application manifest, the EBA file content defines the OSGi application content.
- If an Application-Content header is defined, the EBA file content defines an initial bundle repository from which bundles can be provisioned.

See “Example: OSGi application manifest file” on page 512.

Deployment manifest

A deployment manifest file, `META-INF/DEPLOYMENT.MF`, is created automatically when you import an EBA asset. The deployment manifest file lists, at specific versions, all the bundles and composite bundles that make up the application, including bundles that are determined following dependency analysis. The manifest file is used to ensure that each time an application server starts, the bundles that make up the application are the same.

You can export the current deployment manifest from an EBA asset, then import the deployment manifest into another asset that contains the same application. The target asset then uses the imported manifest instead of the generated manifest. This is useful during application development, when an application is fully tested and moves to a production environment. By importing the deployment manifest from the test environment, you ensure that the bundles and their versions that make up the application in the production environment are exactly the same as the bundles that make up the application in the test environment.

See “OSGi deployment manifest file” on page 514 and Exporting and importing a deployment manifest file.

Enterprise bundle archive installation

OSGi applications in enterprise bundle archive (EBA) files are installed using the business-level application framework.

Installation is a three step process:

1. Import the EBA file as an asset.
2. Create a new business-level application.

   The first two steps can be in either order.
3. Add the asset to the business-level application. The application is configured and associated with the server that it will run on.

During installation, if the EBA file does not contain a deployment manifest, a deployment manifest is generated. If the EBA file contains a deployment manifest, it is checked against the application manifest, and installation continues if the information in the manifests match.

Enterprise bundle archive update

After you import an enterprise bundle archive (EBA) file as an asset, newer versions of the OSGi bundles that it uses might become available. If you want an OSGi application to use a later version of a bundle, you must specify this explicitly by configuring the asset.

The application manifest in an EBA file can define a list of application modules, together with a range of versions for each module. In this way, updates from the minimum to the maximum version of a module are allowed.

The deployment manifest, which is created automatically when you import the EBA file as an asset, specifies the exact version of each module, and ensures that each time an application server starts, the bundles that make up the application are the same. When newer versions of the bundles become available, they are not updated automatically, even if they are within the version range specified in the application manifest.

If an OSGi application requires a newer version of a bundle that has been installed into a configured bundle repository, the EBA asset must be explicitly updated to pull in that newer version. Otherwise, the application continues to use the original deployed bundle version.

You can use the administrative console to list the bundles that make up the application and see the current versions in use. All the bundles except those that are provisioned by the runtime environment are listed. If later versions of any bundles are available in the configured bundle repositories, these versions are also listed, so you can choose to update to one of the later versions.

You can change the version of one or more bundles, then preview whether the changes resolve successfully. You can update a single bundle, or update the whole application. After you preview the update, you can commit or cancel the changes. If you commit the changes, the deployment manifest of the asset is updated. The changes take effect the next time that the business-level application that contains the asset is started.

In a production environment, when you want to ensure that the same versions of bundles that are used for acceptance test are used in production, you can export the appropriate deployment manifest from the test environment to the production environment. When you do this, the application manifest and the deployment manifest are checked to make sure that they contain matching information.

Composite bundles

A composite bundle groups shared bundles together into aggregates. It provides one or more packages at specific versions to an OSGi application. You can also extend a deployed application by adding one or more composite bundles to the composition unit for the application.

A composite bundle is packaged as a composite bundle archive (CBA) file. This file is a compressed archive file with a `.cba` file extension. If the composite bundle is part of an enterprise OSGi application, the CBA file can be directly contained within the enterprise bundle archive (EBA) file for the application, or pulled in by reference from the internal bundle repository or from an external repository that can process composite bundles. A composite bundle can directly contain bundles in its CBA file. It can also include by reference bundles that are hosted alongside the CBA file within the same EBA file, or bundles that are installed in the same bundle repository.

A composite bundle has the following differences from an enterprise OSGi application:

- The bundles that a composite bundle contains or references are defined with exact versions. The bundles in an enterprise OSGi application can be defined with version ranges.
- A composite bundle has a composite bundle manifest, which is a modularity statement that asserts that bundles can be deployed, not that they will resolve. An enterprise OSGi application has an application manifest, which is a provisioning statement.
- A composite bundle can import or export packages. An enterprise OSGi application cannot do this.
- An enterprise OSGi application does not need to fully define its content. When the application is deployed, dependencies are analyzed and additional bundles are provisioned.

There are two main uses for a composite bundle:

- When you want to ensure consistent behavior from a set of shared bundles in an OSGi application, you use a composite bundle to provide that set of bundles to the application. If a required package or service is available at the same version from both a bundle and a composite bundle, the provisioning process selects the package or service from the composite bundle.
- When you want to extend a deployed business-level application that contains an OSGi application, and you do not want to stop the application or modify the underlying EBA asset, you add one or more composite bundles to the composition unit.

  After you import the enterprise bundle archive (EBA) file for your OSGi application as an asset, you can update versions of existing bundles but you cannot add extra bundles to the asset. By adding one or more composite bundles to the composition unit, you can extend a business-level application without having to redevelop and redeploy the underlying OSGi application. When you save the changes to the composition unit, the associated business-level application is updated to use the new configuration. If the business-level application is running, the bundle and configuration updates are applied immediately.

An OSGi application can include a composite bundle just like any other bundle, either directly or by reference. If you want to include the composite bundle by reference, or use the composite bundle to extend a deployed application, the composite bundle must be available in the internal bundle repository or in an external repository that can process composite bundles. If you install a composite bundle in a bundle repository, and the composite bundle includes bundles by reference, you must ensure that the referenced bundles are also available in the same repository. If you use the internal bundle repository, and the composite bundle directly contains bundles, the contained bundles are not listed separately and are only available as part of the composite bundle.

For users who are moving from a previous version:

- In the WebSphere Application Server Version 7 Feature Pack for OSGi Applications and Java Persistence API 2.0, when you add a composite bundle to the internal bundle repository, and that composite bundle directly contains bundles (in compressed files in the root directory of the composite bundle archive file), those bundles are added to the internal bundle repository both as part of the composite bundle and as individually-available bundles. If you later delete the composite bundle from the repository, the individually-available copies of the bundles are not deleted. You might have used this mechanism as a convenient way to upload sets of bundles to the repository.

- In the current version, when you add to the repository a composite bundle that directly contains bundles, those bundles are not also added individually. If you want to add sets of bundles to the internal bundle repository, you package each set as a compressed archive file with a `.zip` file extension, then add the archive file to the repository. The system expands the file, and all the bundles in its root directory are added individually to the repository.

Application bundles, use bundles and provision bundles

Application bundles are instance-specific, and each instance of an application includes its own instance of the bundle. Shared bundles are not instance-specific, and a single instance of a package or service from a shared bundle can be used by many applications. Shared bundles are further sub-divided into use bundles and provision bundles.

Application bundles

Application bundles are bundles that you create specifically for your application. They are instance-specific or *isolated*; that is, they are not intended to be shared. They are referenced in the application manifest in the Application-Content header.

Shared bundles

Shared bundles are not application-specific. A single instance of a package from a shared bundle can be used by many applications. Shared bundles cannot import packages or services from application bundles. Shared bundles must be provided by reference rather than contained directly in an application.

Shared bundles are further subdivided into *use bundles* and *provision bundles*:

Use bundles

A *use bundle* provides at least one package to an application bundle. Use bundles are shared bundles that are referenced in the application manifest in the Use-Bundle header.

By specifying a particular shared bundle as a use bundle, you can control which bundle is provisioned to provide a shared package. For example, if there are two possible providers of a package, bundle A and bundle B, and there is a use bundle statement for bundle A, then bundle A is always provisioned and used.

Provision bundles

A *provision bundle* provides at least one package or service to an application bundle, a use bundle or another provision bundle. Provision bundles are not referenced in the application manifest, and your application does not know how the requirement for each provision bundle is satisfied.

If you have two separate OSGi applications, and you want them to share the same API classes, you can package those classes as a shared bundle then reference that bundle in the Use-Bundle header of both application manifests. For administrators, another benefit of use bundles is that you can monitor and update them using the administrative console or wsadmin commands.

Bundle usage and bundle provisioning terminology

OSGi bundles can be stored in any of the following locations:

- The enterprise bundle archive (EBA) file for the application.
- The internal bundle repository.
- External OSGi bundle repositories.

Application bundles can be stored either in the EBA file or in a repository. *Shared bundles* are stored in a repository (otherwise they cannot be shared).

The process of getting bundles from the repositories is known as *provisioning*. For provisioning purposes, the following terminology is used for bundles:

Referenced bundles

A *referenced bundle* is a bundle that is referenced in the application manifest, and stored in a repository.

Dependency bundles

A *dependency bundle* is a bundle that is not referenced in the application manifest, but that is used by bundles that *are* referenced in the application manifest, or by other dependency bundles.

This is how the terminology for bundle usage (that is, *application*, *use* and *provision bundles*) maps to the terminology for bundle provisioning (that is, *referenced* and *dependency bundles*):

- *Application bundles* that are not directly contained in the EBA file are instance-specific *referenced bundles*.
- *Use bundles* are shared *referenced bundles*.
- *Provision bundles* are shared *dependency bundles*.

Web application bundles

A web application bundle (WAB) is a bundle that contains a web application and that can be deployed in an OSGi container. A WAB is an OSGi bundle version of a web application archive (WAR) file.

WABs are defined in the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification.

You use a WAB in an OSGi application in much the same way that you use a WAR file in a Java enterprise application. For example, you use a WAB to host servlets, static content, or JavaServer Pages (JSPs) that are part of your application.

A WAB contains a manifest file, `META-INF/MANIFEST.MF`. This file contains the same minimum set of manifest headers that any OSGi bundle manifest contains (for example `Bundle-SymbolicName` and `Bundle-ManifestVersion`), and also extra headers that are specific to WABs. You use these headers to describe your web application in OSGi terms, and to specify the support that you want from the OSGi Applications runtime environment. For example, you do not need to package dependencies inside a WAB; you can instead specify these dependencies in a manifest header and let the runtime environment provision them for you. The only required extra header is `Web-ContextPath`. You use this header to specify the default context from which the web content is hosted.

An OSGi WAB that uses Servlet 3.0 metadata can contain EJBs. If your WAB contains EJBs, and you use a deployment descriptor XML file to define your EJB metadata, then the file must be located at `WEB-INF/ejb-jar.xml`. Any IBM binding files must be located in the `WEB-INF` folder; for example, `WEB-INF/ibm-ejb-jar-bnd.xml`.

You can convert an enterprise application that contains only WAR files to an OSGi application packaged as an enterprise bundle archive (EBA) file. When you import the EBA file as an asset, the WAR files are automatically converted to WABs.

Considerations for using web applications and bundle fragments in web application bundles

There are a few specifics to consider when you develop a web-enabled OSGi application, especially if your application uses the Servlet 3.0 or Contexts and Dependency Injection (CDI) capabilities of the Java Platform, Enterprise Edition 6 (Java EE 6) specification.

- "Files in the root directory"
- "Fragment attach order"
- "Class path ordering in web fragments"
- "Web application bundles and CDI"

Files in the root directory

Any files that are in the root directory of a web application can be viewed or downloaded using a browser. To prevent your bundles and bundle fragments from being accessed in this way, you should keep your `.class` files in a subdirectory such as `WEB-INF/classes`, and your `.jar` files in a subdirectory such as `WEB-INF/lib`, rather than in the root of the bundle.

Fragment attach order

You can extend bundles by adding bundle fragments. However you cannot control the order in which bundle fragments are attached.

To support the Java Servlet 3.0 specification, this implementation lets you define a web application through annotations in bundles and bundle fragments, as well as (or instead of) through a `web.xml` file for the application. The configuration for a web application is built by scanning its class path for annotations, and the order of components on the class path depends upon the fragment attach order.

Because you cannot control the order in which bundle fragments are attached, you cannot predict the order of items on the class path. This means that the design for your web application must not rely on bundle fragments being attached in a particular order.

Class path ordering in web fragments

The Java Servlet 3.0 specification lets you use web fragments to define the class path order. For OSGi applications, the bundle class path is the primary mechanism for specifying the ordering of components. If you use the Java Servlet 3.0 ordering capabilities for web fragments, you must ensure that the resulting class path order matches the bundle class path.

Web application bundles and CDI

For Contexts and Dependency Injection (CDI) to work for a directory or Java archive (JAR) file on a bundle class path, you must add the file `META-INF/beans.xml` within that directory or JAR file. For example, consider the following bundle class path:

```
BundleClassPath:WEB-INF/myclasses,WEB-INF/yourclasses.jar,
WEB-INF/hisclasses,WEB-INF/herclasses.jar
```

For CDI to work with a class `WEB-INF/myclasses/a.class`, you need to add the file `WEB-INF/myclasses/META-INF/beans.xml`. Similarly, for CDI to work with a class `WEB-INF/herclasses.jar/b.class`, you need to add the file `WEB-INF/herclasses.jar/META-INF/beans.xml`.

Note: If you convert a web application as described in Converting a web application archive file to an OSGi web application bundle, and the web archive (WAR) file has classes in the `WEB-INF/classes` directory and has a WEB-INF/beans.xml file, then the conversion process automatically adds the file WEB-INF/classes/META-INF/beans.xml to the web application bundle (WAB) file that is hosted.

EJB bundles

An EJB bundle is a bundle that contains EJBs and that can be deployed in an OSGi container. An EJB bundle is an OSGi bundle version of an EJB JAR file.

You use an EJB bundle in an OSGi application in much the same way that you use an EJB JAR file in a Java enterprise application. For example, you use EJBs to implement the business logic of your application.

An EJB bundle contains a manifest file, META-INF/MANIFEST.MF. This file contains the same minimum set of manifest headers that any OSGi bundle manifest contains (for example Bundle-SymbolicName and Bundle-ManifestVersion), plus the Export-EJB header; it is the presence of the Export-EJB header that identifies the bundle as an EJB bundle.

You use the manifest headers to describe your EJB bundle in OSGi terms, and to specify the support that you want from the OSGi Applications runtime environment. For example, you do not need to package dependencies inside an EJB bundle; you can instead specify these dependencies in a manifest header and have the runtime environment provision them for you.

If you use a deployment descriptor XML file to define your EJB metadata, then the file must be located at META-INF/ejb-jar.xml. Any IBM binding files must be located in the META-INF folder; for example, META-INF/ibm-ejb-jar-bnd.xml.

You can convert an existing EJB JAR file to an EJB bundle by adding OSGi metadata to the file; Rational Application Developer provides tooling to help simplify this task.

Bundle and package versioning

Every bundle or export package has a version number in a specific format, and every import package statement has a version range. When you specify a version range, you need to consider current policy and best practices, and requirements for compatibility with both earlier and future versions.

Version numbers

Every bundle or export package has a version number. This version number consists of up to three numbers, in the following format:

```
9.9.9
```

The first number specifies the *major* component of the version number. The second number (if present) specifies the *minor* component of the version number, and the third number (if present) specifies the *micro* component of the version number. If you omit a component level, you also omit the period character "." that comes before that component level. For example, the following three numbers all specify the same version:

```
9
9.0
9.0.0
```

According to the OSGi versioning policy, when a new version of a bundle or export package is not compatible with earlier versions you increment the *major* version number of the bundle or package.

Version ranges

Every import package statement specifies a version range. The OSGi Alliance recommends the following best practice for specifying a version range:

- For the lower boundary of the range, specify the minimum version of the package that the consuming bundle requires.
- For the upper boundary of the range, include any minor version but exclude any increment in the major version of the package.

Version ranges are specified using the following notation:

- A square bracket “[” or “]” means “include the end of the specified range”.
- A round bracket “(” or “)” means “exclude the end of the specified range”.

For example, the following import statement can resolve against any version of a package from version 1.0 up to, but not including, version 2.0.

```
Import-Package: com.myco.a.pkge;version="[1.0,2.0)"
```

A package at version 1.3 is expected to be compatible with a package at version 1.0, and so a bundle that requires at least version 1.0 of a package should work if it is resolved against version 1.3 of that package. However, the bundle might not work if it is resolved against version 2.0 of the package, so version 2.0 and later versions are excluded.

Notes:

- When new methods are added to an interface, the minor version number is incremented for the export package header. A class that implements the previous version of the interface can also work with the new version, but the new version is only made available if explicitly specified. In this case, the bundle importing the package that contains the interface should specify (for example) `"[1.0,1.1)"`.
- The Java Community Process (JCP) does not define OSGi metadata such as package versions when it defines APIs. Currently, the OSGi versions are defined in the OSGi specifications that reference the Java technologies. Future versions of JCP specifications are expected to be compatible with earlier versions, so the OSGi package versioning policy for such future specifications could simply reflect the JCP specification version. This might result in a change of major version of a javax package even when the new version is compatible with the old. For example, the Java Persistence API (JPA) 2.0 is compatible with the earlier version JPA 1.0. You might expect the javax.persistence package version to follow the JCP specification name and use version 2.0. However, it actually follows OSGi versioning policy and uses version 1.1.

  Because JCP specifications always try to maintain compatibility with earlier versions, you should specify only the minimum package version for a javax.* package and leave the upper boundary open. For example, suppose that a future version of WebSphere Application Server has a default JPA provider that implements a version of JPA later than Version 2.0. In this case, the application server can export the javax.persistence package at that later version and still be resolved by applications that are written to work with JPA 1.0 or 2.0. Similarly, the following example import statement can resolve against any version of the javax.persistence package at 1.0 or higher:

  ```
  Import-Package: javax.persistence;version="1.0"
  ```

Manifest files

The metadata for an OSGi application is defined in manifest files. An OSGi bundle contains a bundle manifest; a composite bundle archive (CBA) contains a composite bundle manifest; an enterprise bundle archive (EBA) contains an application manifest; an EBA asset contains a deployment manifest. The deployment manifest is generated automatically when the EBA file is imported as an asset. You create the other manifest files when you create the bundles or application.

Example: OSGi bundle manifest file

An OSGi bundle, which can be a JAR or web application archive (WAR) file, contains a bundle manifest file `META-INF/MANIFEST.MF`. In addition to the headers that can be defined for a non-OSGi JAR or WAR file, the bundle manifest file for an OSGi bundle contains OSGi-specific headers. The metadata that is specified in these headers enables the OSGi Framework to process the modular aspects of the bundle.

Eclipse tools provides convenient editors for the manifest file.

Here is an example bundle manifest file, `META-INF/MANIFEST.MF`; the headers highlighted in bold are mandatory:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
Bundle-SymbolicName: com.sample.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.sample.myservice.Activator
Import-Package:  org.apache.commons.logging;version="1.0.4"
Export-Package:  com.sample.myservice.api;version="1.0.0"
Meta-Persistence: entities/persistence.xml,
  lib/thirdPartyEntities.jar!/META-INF/persistence.xml
Web-ContextPath: /contextRoot
Export-EJB:
Bundle-Blueprint: /blueprint/*.xml
```

The metadata in a bundle manifest file includes the following headers:

Bundle-SymbolicName
A name that identifies the bundle uniquely.

Bundle-Version
This header describes the version of the bundle, and enables multiple versions of a bundle to be active concurrently in the same framework instance.

Bundle-Activator
This header notifies the bundle of lifecycle changes.

Import-Package
This header declares the external dependencies of the bundle that the OSGi Framework uses to resolve the bundle. Specific versions or version ranges for each package can be declared. In this example manifest file, the org.apache.commons.logging package is required at Version 1.0.4 or later.

Export-Package
This header declares the packages that are visible outside the bundle. If a package is not declared in this header, it is visible only within the bundle.

Export-EJB
The presence of this header identifies this bundle as an EJB bundle, and causes any enterprise beans in the bundle to be loaded and run by the EJB container. The value of this header declares the enterprise beans that you want to export as OSGi services.

The Export-EJB header can have any of the following values:

- A single space character: export all enterprise beans in the bundle.
- A comma-separated list of the class names of the enterprise beans that you want to export. If an enterprise bean is not included in this list, it is still loaded and run, but it is not exposed in the OSGi service registry.
- `NONE`: do not export any enterprise beans.

Note:

- Only session enterprise beans that are not stateful are exported.
- The most straightforward way for a client to use an enterprise bean that is exposed as a service is to import the package that contains the EJB business interface. For a client to

be able to import the required package, that package must be declared in the Export-Package header of this bundle, or any other bundle.

If this bundle is a web application bundle, and you are using Version 3.0 of the Java Servlet Specification, the bundle can contain enterprise beans. In this case, if you specify a Web-ContextPath header then any enterprise beans in the bundle are loaded and run by the EJB container, but none are exposed in the OSGi service registry. This configuration is equivalent to specifying "Export-EJB: NONE". To export one or more enterprise beans from a web application bundle into the OSGi service registry, specify an Export-EJB header in addition to the Web-ContextPath header. The value of the Export-EJB header determines which enterprise beans are exposed.

An exported enterprise bean is registered in the OSGi service registry with the following service properties:

- `ejb.name`: the name of the enterprise bean.
- `ejb.type`: the EJB type; the value of this property is either "Stateless" or "Singleton". For more information about EJB types, see Enterprise beans and Developing session beans.
- `service.exported.interfaces`: the enterprise bean is registered with this property only if it has a remote interface; the value is the EJB interface name. This property configures the enterprise bean as being suitable for remote access from outside the application, through Service Component Architecture (SCA) for example. For more information, see the description of the Application-ExportService header in the application manifest file.

Meta-Persistence

If your application uses the Java Persistence API (JPA), and this bundle is a persistence bundle, then the bundle manifest also contains a Meta-Persistence header. For more information, see “JPA and OSGi Applications” on page 520.

This header lists all the locations of `persistence.xml` files in the persistence bundle. When this header is present, the default location, `META-INF/persistence.xml`, is added by default. Therefore, when the `persistence.xml` files are in the default location, the Meta-Persistence header must be present, but its content can be empty (a single space).

Web-ContextPath

The presence of this header identifies this bundle as a web application bundle.

This header specifies the default context from which the web content is hosted.

Bundle-Blueprint

This header specifies the location of the blueprint descriptor files in the bundle; for more information, see Section 121.3.4 of the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification.

Example: OSGi composite bundle manifest file

A composite bundle groups shared bundles together into aggregates. A composite bundle is described in a composite bundle manifest file, `META-INF/COMPOSITEBUNDLE.MF`. This manifest file lists the OSGi bundles that are directly contained in the composite bundle, and the reference bundles that are hosted alongside the composite bundle in the same EBA file, or in the same bundle repository.

A composite bundle provides one or more packages at specific versions to an application. Therefore all the versions in a composite bundle manifest are exact.

Eclipse tools provides convenient editors for the manifest file.

Here is an example composite bundle manifest file, `META-INF/COMPOSITEBUNDLE.MF`:

```
Manifest-Version: 1.0
CompositeBundle-ManifestVersion: 1
Bundle-Name: Blog Application
```

```
Bundle-SymbolicName: com.ibm.ws.osgi.example.Blog
Bundle-Version: 1.0
CompositeBundle-Content:
 com.ibm.ws.osgi.example.blog;version="[1.0,1.0]",
 com.ibm.ws.osgi.example.blog.persistence;version="[1.0,1.0]"
Import-Package: com.ibm.ws.other.pkge;version=1.0.0
Export-Package: com.ibm.ws.osgi.example.blog;version=1.0.0
CompositeBundle-ExportService:
 com.ibm.ws.osgi.example.blog.BloggingService;filter="(blog.type=community)"
CompositeBundle-ImportService:
 com.ibm.ws.osgi.example.auth.UserAuthService
```

The metadata in a composite bundle manifest file includes the following headers:

Manifest-Version
A version number for the manifest format.

CompositeBundle-ManifestVersion
The composite bundle manifest version to which this manifest conforms.

Bundle-Name
A human-readable name of the composite bundle.

If you do not specify a value, the default value is the composite bundle symbolic name.

Bundle-SymbolicName
A name that identifies the composite bundle uniquely. The name follows the same scheme as the Bundle-SymbolicName header in an OSGi bundle. The value must not be the same as the value of the Bundle-SymbolicName of any of the bundles in the composite bundle, as listed in the CompositeBundle-Content header.

Bundle-Version
A version number that identifies the version of the composite bundle uniquely.

CompositeBundle-Content
A list of bundles in the composite bundle. All bundles must be available for deployment and must be contained in the `.cba` file, or exist in an available bundle repository. Bundles must have exact version numbers. If you require the same composite bundle with different versions of its content, you require different versions of the composite bundle, one version for each usage.

Import-Package
A list of packages that the composite bundle wants to import. This list is developed from the import package lists in the individual bundle manifests within the composite bundle:

- If an individual bundle manifest specifies an import package, and the same package is contained in another bundle in the composite bundle, then the composite bundle need not import the package.
- If a bundle in the composite bundle specifies an import package that is not otherwise available within the composite bundle, then the package must be listed as an import package in the composite bundle manifest.

Export-Package
A list of packages that the contents of the composite bundle provides to the shared bundle space.

Note: A composite bundle can export a package from a bundle JAR file that is contained in the composite bundle, provided that package is also listed in the Export-Package header in the bundle manifest file of the contained bundle. However, the attributes and directives that are specified for that package in the Export-Package header of the composite bundle manifest file must **exactly** match those specified in the manifest file of the contained bundle, otherwise the package is not exported.

CompositeBundle-ExportService
A list of service interface names and optional filters that identify services that are present in the

composite bundle and that can be exported for use outside the composite bundle. The interfaces that an exported service implements are usable outside the composite bundle if those interfaces are visible outside the composite bundle.

The format is a comma-separated list of services, in the form of a service interface name, followed by attributes or directives. The CompositeBundle-ExportService header has the following attribute:

filter An OSGi service filter.

CompositeBundle-ImportService
A list of service interface names and optional filters that identify services that the contents of the composite bundle want to use from outside the composite bundle. At least one such service must exist at run time.

The format is a comma-separated list of services, in the form of a service interface name, followed by attributes or directives. The CompositeBundle-ImportService header has the following attribute:

filter An OSGi service filter.

Example: OSGi application manifest file

The OSGi bundles in an enterprise bundle archive (EBA) file share services with other OSGi applications by declaring them in an application manifest file, `META-INF/APPLICATION.MF`. Any external services and references that the OSGi application produces are exposed by declaring them in the manifest, and any external services and references that the application consumes are also declared in the manifest.

The application manifest describes modularity at the application level. It uses configuration by exception, that is, you supply values only when you want to override the default. An EBA file does not have to contain an application manifest file. The default, when no application manifest is declared, is that the application content is the set of OSGi bundles contained in the OSGi application, and no external services or references are produced or consumed.

The application manifest specifies the bundles that form the primary content of the application. A bundle that is listed in the primary content might use a package that is not included in the application, and therefore require other bundles to be pulled in. The application manifest might also specify an allowed version range for some bundles; this range defines the initial version of the application, and the latest version to which it can be upgraded.

Eclipse tools provides convenient editors for the manifest file.

Here is an example application manifest file, `META-INF/APPLICATION.MF`:

```
Manifest-Version: 1.0
Application-ManifestVersion: 1.0
Application-Name: My Club
Application-SymbolicName: com.myclub.app
Application-Version: 1.0
Application-Content:
  com.myclub.api; version=1.0.0,
  com.myclub.persistence; version=1.0.0,
  com.myclub.web; version="[1.2.0,1.2.5)",
  com.myclub.common; version="(1.2.0,2.0.0)"
Application-ImportService: com.myclub.security.authenticationService; filter="(security=strong)"
Application-ExportService: com.myclub.memberService
Use-Bundle: com.clubs.utils; version="[1.0.0,1.1.0)"
```

The metadata in an application manifest file includes the following headers:

Manifest-Version
A version number for the manifest format.

Application-ManifestVersion
The application manifest version to which this manifest conforms.

Application-Name
A human-readable name of the application.

If you do not specify a value, the default value is the application symbolic name.

Application-SymbolicName
A name that identifies the application uniquely. The name follows the same scheme as the Bundle-SymbolicName header in an OSGi bundle. The value must not be the same as the value of the Bundle-SymbolicName header for any bundle or composite bundle in the application, as listed in the Application-Content header.

If you do not specify a value, the default value is the name of the EBA file.

Application-Version
A version number that uniquely identifies the version of the application. The combination of the application symbolic name and version must be unique in an OSGi application runtime environment.

If you do not specify a value but the application name contains an underscore character "_" followed by a valid version value, this value is used. For example, for the application name com.ibm.ws.eba.example_1.2.3.eba, the default value is 1.2.3.

Otherwise, if you do not specify a value, the default value is 0.0.0.

Application-Content
A list of application modules that comprise the primary content of the application. These can be modules that are contained directly in the EBA file, or bundles that are provided by reference, that is, the core bundles to provision for the application.

If you do not specify a value, the default is the modules that are contained directly in the root of the EBA file.

The format is a comma-separated list of module declarations, where each module declaration uses the following format:

```
bundle_symbolic_name;version
```

The version format is the same as that used for OSGi import (for example, in the Import-Package syntax).

The Application-Content header has the following attribute:

version
The version of the module, specified using OSGi syntax for a version range. Typically, you specify the version of the module when the application is written, and the latest version to which the application can be upgraded.

In OSGi syntax for a version range, brackets [ ] mean include the corresponding lower or upper limit, and parentheses ( ) mean exclude the corresponding lower or upper limit. For example, `[1.0.0,2.0.0)` means version 1.0.0 and all later versions, up to, but not including, version 2.0.0.

The Application-Content header defines the important applications that compose the business services, but it does not define the full list of bundles in the application. If a bundle that is listed in the content uses a package that is not included in the application, dependencies are analyzed when the application is deployed and other bundles are provisioned. Any bundles that are provisioned cannot provide services external to the application and cannot have security applied to them. Such bundles are provisioned to the shared bundle space, rather than being provisioned for each isolated application.

Important: If your application contains web application bundles, EJB bundles, or persistence bundles, you **must** list these bundles in the Application-Content header. If you do not list these bundles in the Application-Content header, they are provisioned to the shared bundle space, and they can continue to provide packages, but their contents are not be processed by web, EJB or JPA containers, and they will therefore not function correctly.

Application-ImportService

A *remotable* service can be accessed outside the application. This header contains a list of service interface names, and an optional filter, that identify remotable services that the application wants to consume from outside the application.

If you do not specify this header, no services are imported. If you do not specify a value, the default is no values.

The format is a comma-separated list of services, in the form of a service interface name, followed by the **`filter`** attribute; this attribute specifies an OSGi service filter.

For example:

```
Application-ImportService: com.myclub.security.authenticationService; filter="(security=strong)"
```

Your remotable service must support pass-by-value semantics. To match against services that are imported by the Application-ImportService header, your service references must look for the **`service.imported`** property.This prevents accidental exposure or use of services that only support only local calls and expect pass-by-reference semantics.

Application-ExportService

A *remotable* service can be accessed outside the application. This header contains a list of service interface names, and an optional filter, that identify remotable services that the application provides.

If you do not specify this header, no services are exported. If you do not specify a value, the default is no values.

The format is a comma-separated list of services, in the form of a service interface name, followed by the **`filter`** attribute; this attribute specifies an OSGi service filter.

For example:

```
Application-ExportService: com.myclub.memberService; filter="(level=silver)"
```

Your remotable service must support pass-by-value semantics. To match against services that are exported by the Application-ExportService header, your service must be configured as remotable (that is, registered with the **`service.exported.interfaces`** property).This prevents accidental exposure or use of services that only support only local calls and expect pass-by-reference semantics.

Use-Bundle

A list of bundles or composite bundles to use to satisfy the package dependencies of bundles in the Application-Content list. Each bundle or composite bundle in the Use-Bundle list must provide at least one package to at least one bundle in the Application-Content list. These bundles will be provisioned into the shared bundle space at run time.

Often, you do not require a Use-Bundle header, but there are some situations where it is useful. You can use it to restrict the level at which sharing is possible. For example, you can ensure that an application uses the same bundle for package imports that it was tested with. Alternatively, you can ensure that two applications use the same bundle for package imports. By setting the restriction at application level, the bundle can remain flexible.

OSGi deployment manifest file

A deployment manifest file, `META-INF/DEPLOYMENT.MF`, is created automatically when you import an EBA asset. The deployment manifest file lists, at specific versions, all the bundles and composite bundles that make up the application, including bundles that are determined following dependency analysis. The manifest file is used to ensure that each time an application server starts, the bundles that make up the application are the same.

You can export the current deployment manifest file from an enterprise bundle archive (EBA) asset. You might want to do this to save the information, or to import it into another identical application.

Note: Do not edit an exported manifest file. Use the export and import options only in situations where you can treat the exported file as a “black box”.

Provisioning for OSGi applications

When you import an enterprise bundle archive (EBA) file as an asset, or update an asset to use new bundle versions, or add a composite bundle as an extension to a composition unit, provisioning ensures that all the required OSGi bundles are available. An OSGi application can use bundles from external repositories, bundles from the internal repository, and bundles that are included in an EBA file or a composite bundle archive (CBA) file.

As an OSGi application becomes formalized, the developer creates an application manifest that lists all the bundles that the application uses directly. Of this set of bundles, the developer might choose to package up only newly-created bundles in the EBA file, expecting other bundles to be provisioned from the configured local and remote bundle repositories. Similarly, the developer of a composite bundle creates a composite bundle manifest that lists all the bundles that the composite bundle uses. Of this set of bundles, the developer might choose to package up some bundles to be directly available to the composite bundle, and expect other bundles to be provisioned from the configured bundle repositories.

OSGi bundles can be stored in the following locations:

- The EBA file.
- The CBA file.
- The internal bundle repository.

  A WebSphere Application Server installation has one internal bundle repository. You can add an OSGi bundle or a composite bundle to the internal repository. If a bundle is used by many OSGi applications, consider adding that bundle to the internal repository.
- External bundle repositories.

  You can specify the location of one or more external bundle repositories that contain the bundles that the application requires. Depending on how the external bundle repository is implemented, you might not be able to use it to provision services, or to store composite bundles or bundles referenced by composite bundles.

Provisioning gets the following types of bundle from a repository:

- *Referenced bundle*. This is a bundle that is referenced in the application or composite bundle manifest, and not directly contained in the EBA file or composite bundle.
- *Dependency bundle*. This is a bundle that is not referenced in the application or composite bundle manifest, but is needed by a bundle that *is* referenced in the application or composite bundle manifest. There might be more than one level of dependency for such a bundle. That is, a dependency bundle might itself be dependent on another dependency bundle.

Provisioning occurs for an EBA asset when the OSGi application is initially imported, and when the asset is subsequently updated. The asset is resolved; that is, the locations of the constituent application bundles, at appropriate versions, are determined by using the contents of the EBA file, the internal bundle repository, and the specified set of external bundle repositories. Similarly, provisioning occurs when you add a composite bundle as an extension to a composition unit. The locations of the constituent bundles, at appropriate versions, are determined by using the contents of the composite bundle and the available bundle repositories.

Provisioning also checks for dependencies and locates them from the relevant bundle repositories. Dependencies include imported packages, required bundles, services, and persistence providers. The provisioning process detects service dependencies by checking the <service>, <reference>, and <reference-list> elements in the Blueprint XML files for a bundle. Bundles are scanned for Blueprint XML files when the bundles are added to the internal bundle repository, or when they are provided in an EBA or CBA file. If your application code makes direct programmatic use of OSGi services, provisioning does not detect those service dependencies unless they are also specified in the Blueprint XML files.

If a bundle is referenced in the application manifest with a range of possible versions, provisioning locates a bundle at a version in that range. If more than one version of a specified bundle is available, the latest version in the specified range is selected, unless selecting a later version prevents the application from resolving.

If a required package or service is available at the same version from both a bundle and a composite bundle, the provisioning process selects the package or service from the composite bundle.

If the constituent application bundles of an OSGi application resolve successfully, a deployment manifest is generated. This manifest lists all the bundles that the application requires, with the actual version of each bundle that the OSGi application will use. The deployment manifest includes all the bundles that are declared in the application manifest, and also any dependency bundles.

Note: If the provisioning process establishes that a bundle that is listed in the Use-Bundle header in the application manifest does not provide any packages to bundles listed in the Deployed-Content header, that use bundle is not listed in the Deployed-UseBundle header in the deployment manifest. Also, unless that use bundle is required for another purpose, it is not listed in the Provision-Bundle header in the deployment manifest.

When the constituent application bundles resolve successfully and the configuration is saved, all the referenced bundles and dependency bundles are downloaded from either the internal bundle repository or external bundle repositories and stored locally in the bundle cache. This cache is cell-wide for network deployment configurations, and server-wide for single server configurations. If any constituent bundles cannot be located, the EBA asset does not resolve. Messages show which bundles cannot be located. Before you can continue with the import or update, you must make the relevant bundle available in one of the following ways:

- Add the bundle to the EBA or CBA file.
- Upload the bundle to the internal bundle repository.
- Find the bundle in an external bundle repository, and specify the location of that repository.

When all bundle downloads are complete, you can add the asset to a business-level application, or update the OSGi composition unit, so that the business-level application uses the newer configuration.

OSGi application isolation and sharing

At run time, OSGi applications are isolated from each other, but their dependencies are shared.

In the runtime environment, OSGi applications are isolated from each other:

- OSGi applications cannot share packages.
- Each OSGi application runs in its own isolated OSGi framework with its own service registry.
- OSGi services can be imported and exported between frameworks by setting the appropriate headers in the application manifest.

For each application server that is running one or more OSGi applications, there is also one shared bundle space.

The primary content of an OSGi application runs in the framework of the application. The dependency bundles and the shared bundles in the application run in the shared bundle space.

When an application is started, the bundles that are listed in the deployment manifest of the application are loaded into the runtime environment. The bundles that are listed in the Deployed-Content header in the deployment manifest are loaded into the isolated framework. The bundles that are listed in the Deployed-Use-Bundle and the Provision-Bundle headers in the deployment manifest are loaded into the shared bundle space.

The dependency bundles in the shared bundle space can originate from the internal bundle repository, the external bundle repository, or the content of the application.

However, shared bundles in an application must be provided by reference rather than contained directly in an application. This is because if shared bundles are contained directly in an application, they are not available to other applications when the applications resolve, but are in the shared bundle space at run time, which might cause problems.

The following shows an example application manifest:

```
Manifest-Version: 1.0
Application-ManifestVersion: 1.0
Application-Name: Example Blog
Application-SymbolicName: example.blog.app
Application-Version: 1.0
Application-Content:
 example.blog.api;version="[1.0.0,2.0.0)",
 example.blog;version="[1.0.0,2.0.0)"
```

When dependencies are resolved, the deployed application needs to pull in an additional dependency bundle `blog.required.bundle` at version `1.2.0`. This bundle is available in package `blog.required.package`. This package is connected from the dependency bundle in the shared bundle space to bundles in both isolated application frameworks, as shown in the following diagram.

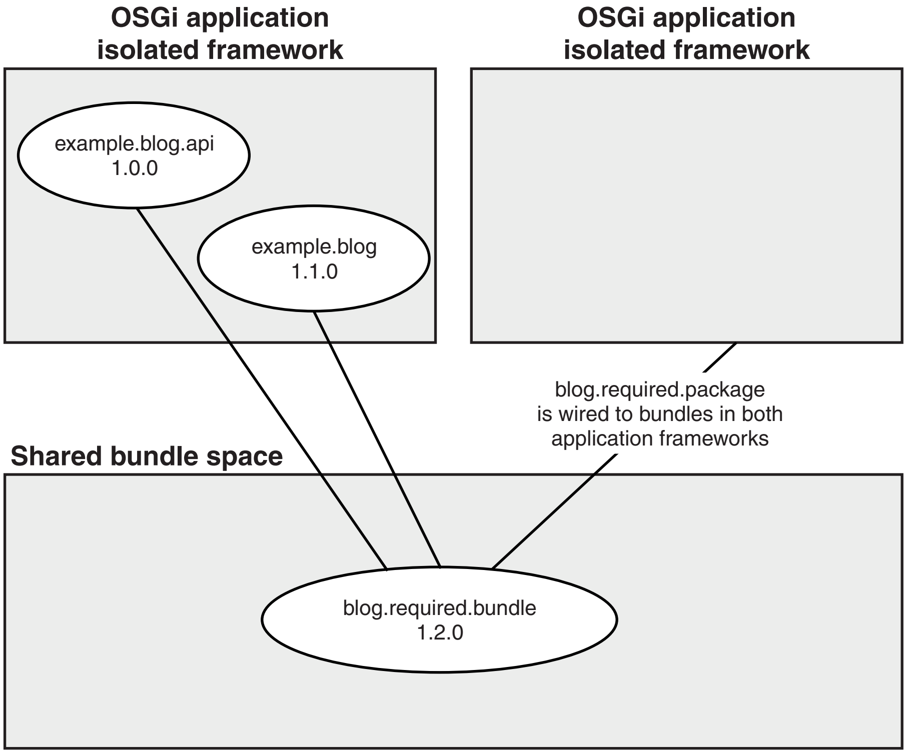

OSGi application isolated framework
OSGi application isolated framework
example.blog.api 1.0.0
example.blog 1.1.0
blog.required.package is wired to bundles in both application frameworks
Shared bundle space
blog.required.bundle 1.2.0

Figure 69. Application frameworks and the shared bundle space

Java 2 security and OSGi Applications

You can use Java 2 security in OSGi applications in a similar way to Java 2 security in Java EE applications. This topic describes the aspects that are specific to using Java 2 security in an OSGi application.

The OSGi specifications allow you to have `permissions.perm` files in the `OSGI-INF` directory of each bundle, so that you can apply fine-grained control to the permissions for each bundle. The OSGi Applications implementation in WebSphere Application Server supports this specification, and also allows you to have a `permissions.perm` file in the `META-INF` directory of the OSGi application, which gives you coarser-grained control of the permissions for the application as a whole.

A `permissions.perm` file is a plain text file that contains comments or single-line permissions in the following form:

```
# Permissions file
( org.osgi.framework.AdminPermission "*" "*" )
( org.osgi.framework.PackagePermission "*" "exportonly,import" )
( org.osgi.framework.ServicePermission "*" "get,register")
( org.osgi.framework.BundlePermission "*" "host,provide,fragment")
```

Relation to Java EE applications and was.policy files

These application-level `permissions.perm` files have a similar function to `was.policy` files in enterprise applications. When you convert an application from Java EE to OSGi, any existing `was.policy` file is converted into a `permissions.perm` file to be used with the OSGi permissions framework.

In the conversion, any codebases specified within the was.policy file are ignored, and all permissions specified are added to the permissions.perm file. This means that all permissions are promoted to the application level. If you need finer granularity, you can modify the file after conversion. In this case, you would remove the required permissions from the resulting permissions.perm file, and move them into permission files within the OSGI-INF directory for each affected bundle.

Default restrictions and permissions

Every OSGi application has the following default restrictions and permissions, whether or not it has a permissions.perm file. You can use a permissions.perm file to add extra restrictions and permissions, or to override default restrictions and permissions.

Default restrictions:

```
("org.osgi.framework.ServicePermission", "org.osgi.service.condpermadmin.ConditionalPermissionAdmin", "*")
("org.osgi.framework.ServicePermission", "org.osgi.service.permissionadmin.PermissionAdmin", "*")
("org.osgi.framework.ServicePermission", "org.osgi.service.framework.CompositeBundleFactory", "*")
("org.osgi.framework.ServicePermission", "org.osgi.framework.hooks.service.*", "*")
("org.osgi.framework.ServicePermission", "org.osgi.service.packageadmin.PackageAdmin", "*")
```

Default permissions:

```
("org.osgi.framework.PackagePermission", "*", "import")
("org.osgi.framework.BundlePermission", "*", "host,provide,fragment")
```

Any OSGi application that has no permissions.perm file also has the following extra permissions:

```
("java.io.FilePermission", "<application_path>/-", "read,write")
("java.io.FilePermission", "<application_configpath>/-", "read")
("java.lang.RuntimePermission", "loadLibrary.*", "*")
("java.lang.RuntimePermission", "queuePrintJob", "*")
("java.net.SocketPermission", "*", "connect")
("java.util.PropertyPermission", "*", "read")
("org.osgi.framework.PackagePermission", "*", "exportonly,import")
("org.osgi.framework.ServicePermission", "*", "get,register")
```

JMS and OSGi Applications

An OSGi application can send and receive Java Message Service (JMS) messages. OSGi applications can use JMS resources that are configured within WebSphere Application Server, in a similar way to using JMS resources with Java EE applications. For OSGi applications, each reference to a JMS resource is declared in a Blueprint XML file.

Each bundle in an OSGi application can contain any number of resource reference declarations in its various Blueprint XML files.

Your OSGi application can bind to any of the following JMS resource types:

- Default messaging JMS queues destinations
- Default messaging JMS topic destinations
- Generic JMS connection factory
- JMS queue connection factory for the JMS provider of WebSphere MQ
- JMS queue destination for WebSphere MQ
- JMS topic connection factory for WebSphere MQ
- JMS topic destination for WebSphere MQ
- Unified JMS connection factory for WebSphere MQ

You bind JMS resource references to your OSGi application when you add the EBA asset to a business-level application.

OSGi applications can access and invoke Enterprise JavaBeans (EJBs) directly. However, you can also configure your OSGi application to send JMS messages to destinations, and configure your enterprise beans or message driven beans (MDBs) to retrieve the messages from those destinations and respond to them.

JPA and OSGi Applications

You can use Java Persistence API (JPA) in OSGi Applications in a similar way to JPA in Java EE applications. This topic describes the differences when you use JPA with persistence bundles in an OSGi application.

OSGi applications use persistence bundles to define the entities and services that other bundles, or the persistence bundle itself, can use.

A persistence bundle is an OSGi bundle that contains one or more persistence descriptors (`persistence.xml` files) and has a Meta-Persistence header in the bundle manifest, `META-INF/MANIFEST.MF`. This header lists all the locations of `persistence.xml` files in the persistence bundle. When this header is present, the default location, `META-INF/persistence.xml`, is added by default. Therefore, when the `persistence.xml` files are in the default location, the Meta-Persistence header must be present, but its content can be empty (a single space).

The following example of a Meta-Persistence header defines a persistence bundle with two persistence descriptors, one in `entities` and one in the nested jar `lib/thirdPartyEntities.jar`. Any `persistence.xml` files that are in the default location can also be used.

```
Meta-Persistence: entities/persistence.xml,
lib/thirdPartyEntities.jar!/META-INF/persistence.xml
```

JPA support for OSGi applications can be container-managed or application-managed. Unmanaged JPA is not supported for OSGi applications.

You can use JPA in web application bundles (WABs) in the same way that you use JPA in web application archives (WAR files), provided that all the persistence logic is contained in the web application.

For more information see the following sections, and the related topic about developing applications that use JPA.

Persistence descriptors

In `persistence.xml` files for an OSGi application, the jta-data-source and non-jta-data-source elements access the data sources through a Java Naming and Directory Interface (JNDI) lookup, a JNDI lookup to the service registry, or through Blueprint. This behavior is instead of the Java Naming and Directory Interface (JNDI) lookups that are used in Java EE applications. The following examples show the syntax for a jta-data-source element:

```
<jta-data-source>
   jndi_name_of_the_data_source
</jta-data-source>
```

```
<jta-data-source>
   osgi:service/javax.sql.DataSource/(osgi.jndi.serviceName=
      jndi_name_of_the_data_source)
</jta-data-source>
```

The following example shows how to configure the JNDI name of a data source when the data source configuration uses a Blueprint resource reference.

```
<jta-data-source>
   blueprint:comp/blueprint_component_name
</jta-data-source>
```

The following example shows how the data source that uses a Blueprint resource reference is configured:

```
<blueprint xmlns="..." ...
   xmlns:rr="http://www.ibm.com/appserver/schemas/8.0/blueprint/
   resourcereference">
   <rr:resource-reference id="blueprint_component_name"
      interface="javax.sql.DataSource"
      filter="(osgi.jndi.service.name=jndi_name_of_the_data_source)">
   <rr:res-auth>Container</rr:res-auth>
   <rr:res-sharing-scope>Shareable</rr:res-sharing-scope>
   </rr:resource-reference>
</blueprint>
```

To specify which version of a persistence provider is accepted, you can use a custom property in the persistence.xml file, as shown in the following example:

```
<persistence-unit name="myPU">
   <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
   <properties>
      <property name="org.apache.aries.jpa.provider.version"
         value="[1.1.0,1.3.0]" />
   </properties>
</persistence-unit>
```

Container-managed JPA

For Blueprint bundles, you can inject persistence contexts and units by using annotations or a blueprint extension.

You can inject persistence contexts and units by using @PersistenceUnit and @PersistenceContext annotations on a blueprint bean. Injection through annotation is supported only for blueprint-managed components.

The Apache Aries JPA container context bundle provides the following Blueprint namespace for dependency injection of managed JPA resources:

http://aries.apache.org/xmlns/jpa/v1.0.0.

You can inject persistence contexts and units by using Blueprint tags from this namespace. For example:

```
<blueprint xmlns:jpa="http://aries.apache.org/xmlns/jpa/v1.0.0" ...>
   ...
   <bean name="myPersistenceBean" class="...">
      <jpa:unit property="emf" unitname="myPU" />
   </bean>
   ...
</blueprint>
```

Application-managed JPA

The OSGi JPA specification requires that an EntityManagerFactory instance is made available in the service registry for each persistence unit that is defined in the persistence descriptors.

The EntityManagerFactory service has three notable service properties:

osgi.unit.name
The name of the persistence unit for the EntityManagerFactory service.

osgi.unit.version
The version of the persistence bundle.

osgi.unit.provider
The name of the persistence provider implementation class for the EntityManagerFactory service.

Transactions and OSGi Applications

You can use transactions in OSGi Applications in a similar way to transactions in Java EE applications. This topic describes the differences when you use transactions with an OSGi application.

You configure transactions by specifying one or more <transaction> elements. The <transaction> element is defined in a Blueprint namespace extension, http://aries.apache.org/xmlns/transactions/v1.0.0. Therefore, you must specify a namespace prefix for this extension in the <blueprint> element. In the examples that follow, the prefix "tx" is used; this prefix is specified as follows:

```
<blueprint
  xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0">
```

You configure transactions at the component level, for example for a bean. You must specify both method and value attributes in the transaction element. Valid values are those that are defined by Java EE, that is, `Required`, `RequiresNew`, `NotSupported`, `Mandatory`, `Supports` or `Never`. For example:

```
<bean ...>
   <tx:transaction method="updateOrder" value="Required" />
</bean>
```

You can specify a list of methods where each method is separated by a space or a comma. For example:

```
<bean ...>
   <tx:transaction method="updateOrder remove" value="Required" />
</bean>
```

You can use an asterisk character (*) as a wildcard in method names. You can use the wildcard character anywhere in a method name and you can use it multiple times. For example:

```
<bean ...>
   <tx:transaction method="update*Ord* remove" value="Required" />
</bean>
```

You can specify multiple method configurations in the same component. For example:

```
<bean ...>
   <tx:transaction method="update*Ord* remove" value="Required" />
   <tx:transaction method="recordStatus" value="RequiresNew" />
</bean>
```

Wildcard matching and selection behavior is determined by the following rules:

1. If more than one transaction element matches a method name, the elements with the least wildcard characters are selected. For example, to match the method updateOrder, the transaction element with the method attribute update* is selected in preference to one with the method attribute update*Ord*.
2. If more than one transaction element still matches, the elements with the longest method attribute are selected. For example, to match the method updateOrder, the transaction element with the method attribute updateOrd* is selected in preference to one with the method attribute update*.
3. If more than one transaction element still matches, an `IllegalStateException` exception is generated.

Transactions and Service Component Architecture (SCA)

You can declare transaction policy at the services level to describe the transactional contract that the service provider offers to the consumer, following the SCA interaction intents model for transactions.

SCA defines two mutually exclusive interaction intents for transactions:

- sca:propagatesTransaction

A service with a policy sca:propagatesTransaction indicates that the service will join any transaction that its requester propagates. A service with the following policy indicates that an SCA service will use a transaction if it is supplied. This configuration is also valid for a transaction with a value of Required on a component implementation.

```
<sca:service name="Service1" requires="sca:propagatesTransaction">
</sca:service>
```

- sca:suspendsTransaction

  A service with a policy sca:suspendsTransaction indicates that the service will not join any transaction that its requester propagates. The component that provides the service might or might not run under its own transaction. A service with the following policy indicates that the service will not use a transaction if it is supplied. This configuration is also valid for a transaction with a value of RequiresNew or NotSupported on a component implementation.

```
<sca:service name="Service1" requires="sca:suspendsTransaction">
</sca:service>
```

For an OSGi application, you declare the interaction intents for services in a module Blueprint file. When you write the Blueprint XML, you must consider whether any remote services should allow the caller to provide a transaction, and whether any service you call remotely should be part of your transaction. If you want your application to be able to participate in a transaction from another application, set the interaction intent to `propagatesTransaction`. However, if your application should run in its own transaction or no transaction, set the property to `suspendsTransaction`. The default value is `suspendsTransaction`, so this value is used if no other value is specified.

The interaction intents for services that are declared in a module Blueprint must be consistent with the transaction definitions in a component. The following table shows the mapping between these two properties:

Transaction definition	Interaction intent
Required	propagatesTransaction
Mandatory	propagatesTransaction
RequiresNew	suspendsTransaction
Supports	propagatesTransaction
NotSupported	suspendsTransaction
Never	suspendsTransaction

The interaction intent is associated with a service entry in the standard OSGi manner: by recording the intent as a service property in the service registry. The OSGi intent property **`service.exported.intents`** is used. This property is defined in the “Remote Services” chapter of the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification.

Services that are implemented by transactional components declare their interaction intent value in the **`service.exported.intents`** property. For example, a service in an OSGi application that is backed by a component with a transaction value of `Required` is registered with the property `service.exported.intents=propagatesTransaction`. This is equivalent to the following module Blueprint service definition:

```
<service ref="componentImplementation"
   interface="com.xyz.MyTransactionalServiceInterface">
   <service-properties>
      <entry key="service.exported.intents" value="propagatesTransaction"/>
   </service-properties>
</service>
```

SCA references specify their transactional requirements on a service by using the same intents:

```
<sca:reference name="Reference1" requires="sca:propagatesTransaction">
</sca:reference>
```

OSGi references specify their transactional requirements on a service by filtering on the OSGi service intents property. The following example code searches for an OSGi service that will participate in a global transaction:

```
<reference id="transactionalService"
   interface="com.xyz.MyTransactionalServiceInterface"
   filter ="(service.exported.intents=propagatesTransaction)"/>
```

The default transaction policy is `Required`.

Handling exceptions

The following table shows how exceptions are handled for transactions in OSGi Applications.

Table 40. Exception handling for transactions in OSGi Applications

Transaction scope	Transaction strategies	Exception generated	Container action	Client view
Client-initiated transaction. The client starts a transaction and propagates it to the bean.	Required Mandatory Supports	Declared exception	Regenerate the declared exception.	The client receives an exception. The client transaction is not affected.
		All other exceptions and errors	Log the exception or error. Mark the transaction for rollback. Regenerate the exception or error.	The client receives an exception. The client transaction is marked for rollback.
Container-managed transaction. A transaction is started before the bean is invoked and ends when the method completes.	Required RequiresNew	Declared exception	Attempt to either commit or roll back the transaction, and regenerate the declared exception.	The client receives an exception. The client transaction is not affected.
		All other exceptions and errors	Log the exception or error. Mark the transaction for rollback. Regenerate the exception or error.	The client receives an exception. The client transaction is not affected.
The bean is not part of a transaction. Any client transaction is not propagated to the bean, and no new transaction is started.	Never NotSupported Supports	Declared exception	Regenerate the declared exception.	The client receives an exception. Any client transaction is not affected.
		All other exceptions and errors	Regenerate the exception.	The client receives an exception. Any client transaction is not affected.

Local transactions

In WebSphere Application Server, Blueprint components always run in a transaction. If you do not configure a global transaction, by using the transaction Blueprint namespace, all methods run in their own local transaction. For more information, see Local transaction containment (LTC).

This local transaction is application-managed with default behaviour. That is, the Resolver attribute of the transaction is set to `Application` and the Unresolved action attribute is set to `Rollback`. The transaction is set to roll back any uncommitted changes. Therefore, any transactional work, such as a database update, that runs in a method of a Blueprint bean, and that is not committed when the method returns, is rolled back, and therefore discarded.

Bean security and OSGi applications

You can configure bean security so that the methods of the bean can be accessed only by users that are assigned a specified role.

You configure security by defining one or more <access-constraint> elements, inside the <bean> element for the bean whose security you want to configure, in the Blueprint XML file of your OSGi application. If you do not define an <access-constraint> element, the bean is not secured; this means that all the methods of the bean can be accessed by any user.

For the bean security configuration to be effective, application security must be enabled in WebSphere Application Server.

You can configure bean security at the bean level and at the method level. If you configure security at both bean level and at method level, the method-level configuration takes precedence.

The <access-constraint> element is defined in a Blueprint namespace extension, http://www.ibm.com/appserver/schemas/blueprint/security/v1.0.0. Therefore, you must specify a namespace prefix for this extension in the <blueprint> element. In the examples that follow, the prefix "sec" is used; this prefix is specified as follows:

```
<blueprint
  xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:sec="http://www.ibm.com/appserver/schemas/blueprint/security/v1.0.0">
```

Configuring bean-level security

To configure security at the bean level, include a role attribute in the <access-constraint> element definition. The methods of the bean can be accessed only by users that are assigned the specified role.

In the following example, the methods of the secureBean1 bean are accessible only by users that are assigned the role called "ROLE1":

```
<bean
  id="secureBean1"
  class="com.ibm.ws.eba.wab.componenttest.blueprint.secure.BlueprintSecureServiceImpl">
  <sec:access-constraint role="ROLE1" />
</bean>
```

Configuring method-level security

To configure security at the method level, include, in addition to the role attribute, a method attribute. The specified method can be accessed only by users that are assigned the specified role.

In the following example, the getPrice method of the secureBean1 bean is accessible only by users that are assigned ROLE1. All the other methods of the bean can be accessed by any user.

```
<bean
  id="secureBean1"
  class="com.ibm.ws.eba.wab.componenttest.blueprint.secure.BlueprintSecureServiceImpl">
  <sec:access-constraint method="getPrice" role="ROLE1" />
</bean>
```

In the following example, the getPrice method is accessible only by users that are assigned ROLE1, and the printReport method is accessible only by users that are assigned ROLE2. All the other methods can be accessed by users that are assigned ROLE3.

```
<bean
  id="secureBean1"
  class="com.ibm.ws.eba.wab.componenttest.blueprint.secure.BlueprintSecureServiceImpl">
  <sec:access-constraint method="getPrice" role="ROLE1" />
  <sec:access-constraint method="printReport" role="ROLE2" />
  <sec:access-constraint role="ROLE3" />
</bean>
```

Assigning roles

To assign a role to users or to groups, use the Security role to user or group mapping panel in the WebSphere Application Server administrative console.

Enterprise JavaBeans and OSGi Applications

An OSGi application can contain Enterprise JavaBeans (EJBs). OSGi applications can access and invoke an enterprise bean directly.

Support is provided for EJB 3.x style enterprise beans, and all EJB functionality works the same in an OSGi application as in a J2EE enterprise application. EJB 2.1 enterprise beans are not supported.

The enterprise beans in your OSGi bundles can be developed from scratch, or you can include existing EJB assets and migrate them to use OSGi modularity with minimal code changes. Stateful, stateless, and singleton enterprise beans are supported. Your OSGi application can also contain message-driven beans (MDBs). You deploy and configure an OSGi application that contains enterprise beans in a similar way to deploying and configuring a J2EE enterprise application, by using wsadmin or the administrative console.

You convert an enterprise application to an OSGi application by completing manual tasks that convert the enterprise archive (EAR) file to an enterprise bundle archive (EBA) file.

The presence of an Export-EJB header in the OSGi bundle manifest file identifies a bundle as an EJB bundle, and causes any enterprise beans in the bundle to be loaded and run by the EJB container. The value of this header declares the enterprise beans that you want to export as OSGi services; for more information, see the description of the Export-EJB header.

You can use the WebSphere Application Server administrative console, or wsadmin scripting commands, to complete the following JNDI binding operations for the enterprise beans in your OSGi applications:

- Provide EJB JNDI names.
- Map EJB references.
- Map EJB resource references to resources.
- Bind EJB message destination references to administered objects..
- Bind listeners for message-driven beans.

Chapter 19. Portlet applications

This page provides a starting point for finding information about portlet applications, which are special reusable Java servlets that appear as defined regions on portal pages. Portlets provide access to many different applications, services, and web content.

Portlet container

Portlets

Portlets are reusable web modules that provide access to Web-based content, applications, and other resources. Portlets can run on the application server because it has an embedded JSR 286 Portlet container. The JSR 286 API provides backwards compatibility. You can assemble portlets into a larger portal page, with multiple instances of the same portlet displaying different data for each user.

From a user's perspective, a portlet is a window on a portal site that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable web modules that are designed to run inside a portlet container of any portal framework. You can either create your own portlets or select portlets from a catalog of third-party portlets.

Each portlet on the page is responsible for providing its output in the form of markup fragments to be integrated into the portal page. The portal is responsible for providing the markup surrounding each portlet. In HTML, for example, the portal can provide markup that gives each portlet a title bar with minimize, maximize, help, and edit icons.

You can also include portlets as fragments into servlets or JavaServer Pages files. This provides better communication between portlets and the Java Platform, Enterprise Edition (Java EE) web technologies provided by the application server.

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

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

Also if you use Rational Application Developer version 6 to create portlets, note that portlets created by using the Struts Portlet Framework are not supported on WebSphere Application Server.

Portlet applications

If the portlet application is a valid web application written to the Java Portlet API, the portlet application can operate on both the Portal Server and the WebSphere Application Server without requiring any changes. JSR 168 and JSR 286 compliant portlet applications must not use extended services provided by WebSphere Portal to operate on the WebSphere Application Server.

Portlet filters

Since the release of JSR 286: Portlet Specification 2.0, it is possible to intercept and manipulate the request and response before they are delivered to a portlet in a given phase. Using *Portlet filters* you can block the rendering of a portlet if a specific condition occurs. Also, you can use portlet filters to decorate a request and a response within a wrapper to modify the behavior of the portlet.

Portlet filter usage

To use the portlet filter feature, you must first complete the following actions:

- Implement one or more of the following interfaces of the `javax.portlet.filter` `package`:
  - `RenderFilter`
  - `ActionFilter`
  - `ResourceFilter`
  - `EventFilter`

  You must implement the appropriate filter based on the method or phase of the portlet that you want to filter.
- Register the filter within the `portlet.xml` file for the portlets in your web application.

The following sample code illustrates a portlet filter to screen the `processAction` method of a portlet:

```
package my.pkg;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.PortletException;
import javax.portlet.filter.ActionFilter;
import javax.portlet.filter.FilterChain;
import javax.portlet.filter.FilterConfig;

public class MyPortletFilter implements ActionFilter {
     public void init(FilterConfig config) throws PortletException {
          String myInitParameter = config.getInitParameter("myInitParameter");
          // ...
     }

     public void doFilter(ActionRequest request, ActionResponse response,
               FilterChain chain) throws IOException, PortletException {
          preProcess(request, response);
          chain.doFilter(request, response);
          postProcess (request, response);
     }

     private void preProcess(ActionRequest request, ActionResponse response) {
          //For example, create a javax.portlet.filter.PortletRequestWrapper here
     }

     public void destroy() {
          // free resources
     }
}
```

The following sample code illustrates how you can declare the previous portlet filter in the `portlet.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
version="2.0" id="demo_app_id">
     <portlet >
          <portlet-name>MyPortlet1</portlet-name>
          <!-- [...] -->
     </portlet>

     <portlet >
          <portlet-name>MyPortlet2</portlet-name>
          <!-- [...] -->
     </portlet>

     <filter>
          <filter-name>PortletFilter</filter-name>
          <filter-class>my.pkg .MyPortletFilter</filter-class>
          <lifecycle>ACTION_PHASE</lifecycle>
          <init-param>
               <name>myInitParameter</name>
               <value>myValue</value>
          </init-param>
     </filter>

     <filter-mapping>
          <filter-name>PortletFilter</filter-name>
          <portlet-name>MyPortlet1</portlet-name>
          <portlet-name>MyPortlet2</portlet-name>
     </filter-mapping>

</portlet-app>
```

If you implement the `RenderFilter` interface, for example, add the `<lifecycle>RENDER_PHASE</lifecycle>` code to the filter section. This addition is analogous to the other filter interfaces. The following values are valid for the `<lifecycle>` parameter:

- `RESOURCE_PHASE`
- `RENDER_PHASE`
- `EVENT_PHASE`
- `ACTION_PHASE`

Global portlet filters

The portlet container for WebSphere Application Server extends the portlet filter feature, which is provided by JSR 286, to allow you to register filters on a global level. These global filters apply to all portlets that are running within the portlet container, including both plain portlets and console modules.

To use global portlet filters, add the following code to the root folder of your Java archive (JAR) file or in the `WEB-INF` directory of your web application and name it `plugin.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin id="portlet-filter-config" name="WS_Server" provider-name="IBM"
version="1.0.0">

     <extension point="com.ibm.ws.portletcontainer.portlet-filter-config">

          <portlet-filter-config class-name="my.pkg.PortletFilter" order="22000">
               <description>Description of global PortletFilterImpl</description>
                    <lifecycle>ACTION_PHASE</lifecycle>
                    <lifecycle>EVENT_PHASE</lifecycle>
                    <lifecycle>RENDER_PHASE</lifecycle>
                    <lifecycle>RESOURCE_PHASE</lifecycle>
                    <init-param>
                         <name>MyInitParam1</name>
                         <value>MyInitValue1</value>
                    </init-param>
                    <init-param>
                         <name>MyInitParam2</name>
                         <value>MyInitValue2</value>
                    </init-param>
          </portlet-filter-config>
     </extension>
</plugin>
```

The `order` attribute of the `portlet-filter-config` element defines when in the filter chain to run the filter. The higher the value, the later the filter runs in the filter chain. Global filters are triggered before local portlet filters.

Note:

- Because global portlet filters are applied to all portlets within the container, when you call the `FilterConfig#getPortletContext` method within the `init` method the return value is null for global portlet filters.
- Do not confuse this feature with the portlet document filters for WebSphere Application Server. Those portlet document filters are, technically speaking, servlet filters that you can apply to rendered output only. For more information about the portlet document filters, see the documentation about converting portlet fragments to an HTML document.
- Because global portlet filters affect all portlets running in the given portlet container, the console modules that are contained in the Integrated Solutions Console are also filtered. It is important to test your filter implementation for undesired side effects on console modules or portlets. One approach is to test by checking the context path of the request in your filter logic.

Portlet container

The *portlet container* is the runtime environment for portlets using the JSR 286 Portlet specification, in which portlets are instantiated, used, and finally destroyed. The JSR 286 Portlet application programming interface (API) provides standard interfaces for portlets and backwards compatibility for JSR 168 portlets. Portlets that are based on this JSR 286 Portlet Specification are referred to as *standard portlets*.

The PortletServingServlet servlet provides a simple portal framework, which builds on top of the portlet container. This servlet registers itself with each web application that contains portlets. You can use the PortletServingServlet servlet to directly render a portlet into a full browser page by a URL request and invoke each portlet by its context root and name. For more information, see Portlet Uniform Resource Locator (URL) addressability. Also, you can use the URL addressability feature to include remote portlet content using the remote request dispatcher (RRD).

If you want to aggregate multiple portlets on a page, you can use the dedicated aggregation tag library, which is based on the PortletServingServlet servlet. For more information, see Portlet aggregation using JavaServer Pages for additional information. For coordination between portlets on a given page, the aggregation tag library supports the use of the Public Render Parameters, as specified by the JSR 286 Portlet Specification.

The portal framework, which is provided with the PortletServingServlet servlet, enables you to render only one portlet at a time. Thus, only a subset of the optional features in the JSR 286 Portal Specification are supported. To determine which optional features are supported, see Supported optional features of the JSR 286 Portlet Specification.

Attention: The brokering of events between portlets is out of the scope of the specification and is not handled by the PortletServingServlet servlet. For full portlet coordination support, you can deploy the portlets on comprehensive portal products, such as the WebSphere Portal Server.

You can disable the PortletServingServlet servlet in an extended portlet deployment descriptor called the `ibm-portlet-ext.xmi` file. For more information, see Example: Configuring the extended portlet deployment descriptor to disable PortletServingServlet.

Chapter 20. Service integration

This page provides a starting point for finding information about service integration.

Service integration provides asynchronous messaging services. In asynchronous messaging, producing applications do not send messages directly to consuming applications. Instead, they send messages to destinations. Consuming applications receive messages from these destinations. A producing application can send a message and then continue processing without waiting until a consuming application receives the message. If necessary, the destination stores the message until the consuming application is ready to receive it.

Service integration technologies

Service integration is a set of technologies that provides asynchronous messaging services. Use this topic to learn about the technologies on which WebSphere Application Server service integration applications are developed and implemented.

Service integration buses and bus members

Application servers or clusters of application servers in a WebSphere Application Server cell can cooperate to provide asynchronous messaging services. Service integration provides asynchronous messaging services, and a group of servers or clusters that cooperate in this way is called a service integration bus. The application servers or server clusters in a bus are known as bus members. You can also add bus members that are WebSphere MQ servers; service integration uses these bus members to write messages to, and read messages from, WebSphere MQ queues.

Different service integration buses can, if required, be connected. This allows applications that use one bus (the local bus) to send messages to destinations in another bus (a foreign bus). Note, though, that applications cannot receive messages from destinations in a foreign bus.

Messaging engines

Each service integration server or cluster bus member contains a component called a messaging engine that processes messaging send and receive requests and that can host destinations. To host queue-type destinations, the messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them, or preserve messages in case the messaging engine fails.

Messaging providers

WebSphere Application Server applications invoke asynchronous messaging services by using the Java Messaging Service (JMS) application programming interface (API) to interface to a messaging provider. WebSphere Application Server supports a variety of JMS messaging providers, including service integration (which is the default messaging provider) and WebSphere MQ as an external JMS messaging provider.

Service integration buses

A *service integration bus* is a group of one or more application servers or server clusters in a WebSphere Application Server cell that cooperate to provide asynchronous messaging services. The application servers or server clusters in a bus are known as bus members. In the simplest case, a service integration bus consists of a single bus member, which is one application server.

Usually, a cell requires only one bus, but a cell can contain any number of buses. The server component that enables a bus to send and receive messages is a messaging engine.

A service integration bus provides the following capabilities:

- Any application can exchange messages with any other application by using a *destination* to which one application sends, and from which the other application receives.

- A message-producing application, that is, a *producer*, can produce messages for a destination regardless of which messaging engine the producer uses to connect to the bus.
- A message-consuming application, that is, a *consumer*, can consume messages from a destination (whenever that destination is available) regardless of which messaging engine the consumer uses to connect to the bus.

Different service integration buses can, if required, be connected. This allows applications that use one bus (the local bus) to send messages to destinations in another bus (a foreign bus). Note, though, that applications cannot receive messages from destinations in a foreign bus.

An application can connect to more than one bus. For example, although an application cannot receive messages from destinations in a foreign bus, if the application connects to that bus, the bus becomes a local bus and then the application can receive messages.

For example, in the following diagram, the application can send messages to destination A and destination B, but it cannot receive messages from destination B.

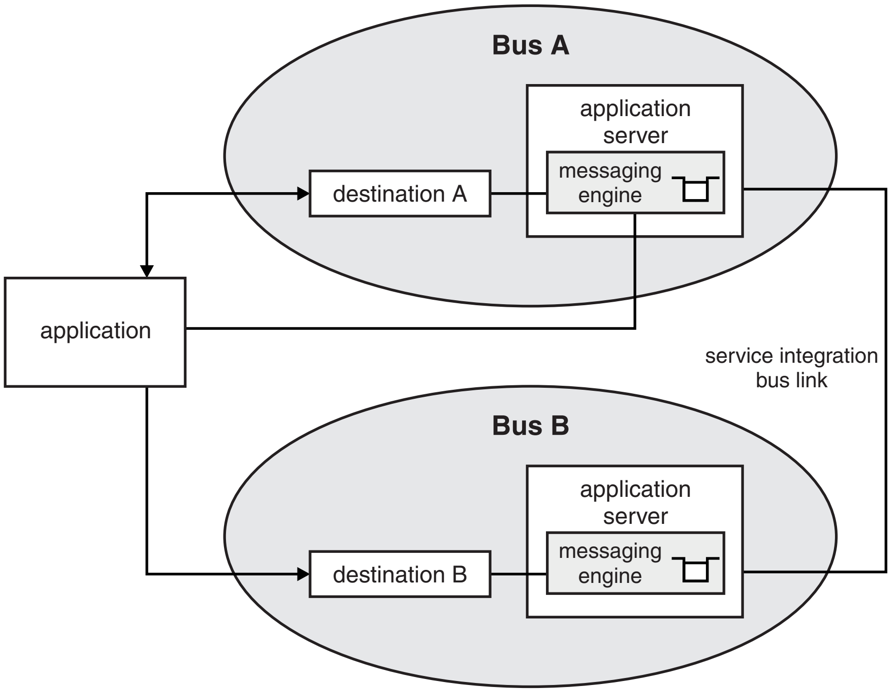


Figure 70. An application that is connected to bus A

In the following diagram, the application can send messages to, and receive messages from, destination A and destination B.

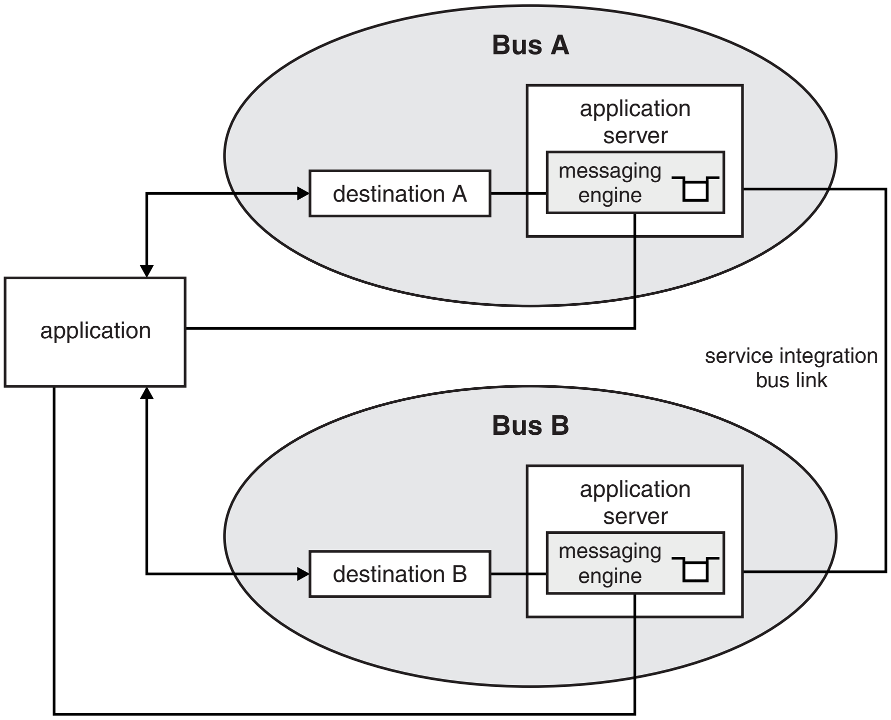


Figure 71. An application that is connected to bus A and bus B

A service integration bus comprises a *SIB Service*, which is available on each application server in the WebSphere Application Server environment. By default, the SIB Service is disabled. This means that when a server starts it cannot undertake any messaging. The SIB Service is enabled automatically when you add a server to a service integration bus. You can choose to disable the service again by configuring the server.

A service integration bus supports asynchronous messaging, that is, a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to the message. Asynchronous messaging is possible regardless of whether the consuming application is running, or whether the destination is available. Also, point-to-point and publish/subscribe messaging are supported.

After an application connects to the bus, the bus behaves as a single logical entity and the connected application does not have to be aware of the bus topology. In many cases, connecting to the bus and defining bus resources is handled by an application programming interface (API) abstraction, for example the administered JMS connection factory and JMS destination objects.

The service integration bus is sometimes referred to as the *messaging bus* if it provides the messaging system for JMS applications that use the default messaging provider.

Many scenarios require a simple bus topology, for example, a single server. If you add multiple servers to a single bus, you increase the number of connection points for applications to use. Servers, however, do not have to be bus members to connect to a bus. In more complex bus topologies, multiple buses are configured, and can be interconnected to form complex networks. An enterprise might deploy multiple interconnected buses for organizational reasons. For example, an enterprise with several independent departments might want separately administered buses in each location.

Bus members

The members of a service integration bus can be application servers, server clusters, or WebSphere MQ servers. Bus members that are application servers or server clusters contain messaging engines, which are the application server components that provide asynchronous messaging services. Bus members that are WebSphere MQ servers provide a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager.

To use a service integration bus, you must add at least one application server or server cluster as a bus member. You can also add bus members that are WebSphere MQ servers; service integration uses these bus members to write messages to, and read messages from, WebSphere MQ queues.

When you add an application server or a server cluster as a bus member, a messaging engine for that bus member is created automatically. If the bus member is an application server, it can have only one messaging engine. To host queue-type destinations, the messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them, or preserve messages in case the messaging engine fails. If the bus member is a WebSphere MQ server, it does not have a messaging engine, but it lets you access WebSphere MQ queues directly from WebSphere MQ queue managers and (for WebSphere MQ for z/OS) queue-sharing groups.

A WebSphere Application Server application does not have to be running on a service integration bus member to use its messaging services. If necessary, WebSphere Application Server automatically provides a connection to a suitable bus member.

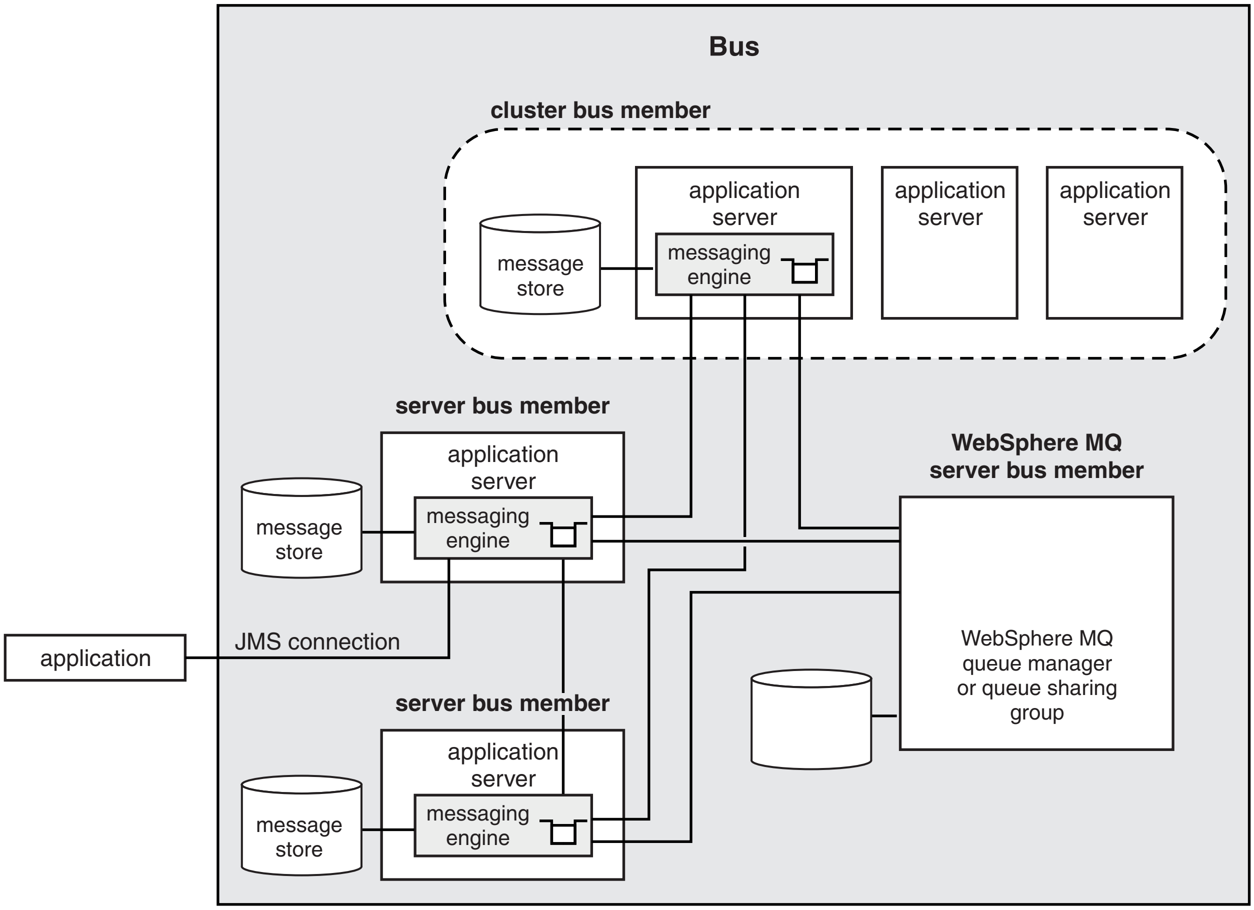


Figure 72. Bus members

If required, you can remove members from a bus. However, this action deletes any messaging engines that are associated with a bus member, including knowledge of any messages held by the message store for those messaging engines. Therefore, you must plan this action carefully.

When a bus member is deleted, the data source associated with this bus member is not automatically deleted, in case you are also using the data source for some other purpose. This also applies to bus members created using the default data source: the data source is not automatically deleted and you must remove it manually. You need not remove the default data sources because they use a universal unique identifier (UUID) in the name of the Apache Derby database. However, you might choose to delete the data source to avoid wasting disk space.

Messaging engines

Each service integration server or cluster bus member contains a component called a messaging engine that processes messaging send and receive requests and that can host destinations.

When you add an application server or a server cluster as a bus member, a messaging engine is automatically created for this new member. If you add the same server as a member of multiple buses, the server is associated with multiple messaging engines (one messaging engine for each bus). If the bus member is a WebSphere MQ server, it does not have a messaging engine, but it lets you access WebSphere MQ queues directly from WebSphere MQ queue managers and (for WebSphere MQ for z/OS) queue-sharing groups.

To host queue-type destinations, the messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them, or preserve messages in case the messaging engine fails. There are two types of message store, file store and data store. For further information, see Administering message stores.

Messaging engines are given a name which is based on the name of the bus member. Each messaging engine also has a universal unique identifier (UUID) that provides a unique identity for the messaging engine.

Note: If you delete and recreate a messaging engine, it will have a different UUID and will not be recognized by the bus as the same engine, even though it might have the same name. For example, the recreated messaging engine will not be able to access the message store that the earlier instance used. If you accidentally delete a messaging engine configuration, and save the updated incorrect configuration, you must restore the configuration from a previous configuration backup.

Mechanisms for stopping messaging engines

There are several different mechanisms that you can use to stop messaging engines. You can also specify two different degrees of urgency: `immediate` and `force`. Stopping a messaging engine prevents it from sending messages.

You can stop messaging engines by:

- Using the administrative console to stop the messaging engine
- Using The JMX **`stop`** command
- Using the **`stopServer`** command to stop the application server that hosts the messaging engine

You can stop a messaging engine in two modes: immediate and force.

Immediate

In immediate mode, the messaging engine is stopped on completion of all the messaging operations that are current at the time of the stop request. No notification is sent to the application to indicate that the messaging engine is stopping. After a `stop` command has been issued, the messaging engine does not allow new operations to be started.

For each existing connection, the messaging engine waits for the current operation to complete, unless the operation blocks processing in the messaging engine, such as a receive operation. In this case, the operation is interrupted. Asynchronous consumers are allowed to complete even though they might take an arbitrary amount of time to process the current message. The messaging engine then backs out of active transactions and disallows further operations on that connection. When all connections are in this invalidated state, the messaging engine stops.

Force In force mode, the messaging engine is stopped so that any current transactions are pre-empted and applications are forcefully disconnected.

Force mode is like immediate mode, except that stopping the messaging engine interrupts messaging operations on application threads that are taking place at the time that the stop command is issued. Rather than allowing existing messaging operations to complete, the messaging engine interrupts them and then disallows further operations. When all connections are in this state, the messaging engine stops.

Force mode completes the shutdown of the messaging engine as fast as possible. A subsequent restart of the messaging engine might take longer than if it had been stopped using immediate mode, because more recovery actions are needed. For example, force mode stop can leave messages with indoubt transactions and you must deal with these messages as described in Resolving indoubt transactions.

You can escalate an immediate stop that is taking too long to force a stop.

The following stop modes are possible for the different stop mechanisms:

Table 41. Comparison of stop mechanisms. The first column lists the mechanisms for stopping the messages. The second column states whether the immediate mode is used as the stopping mechanism. The third column states whether the force mode is used as the stopping mechanism.

Stop mechanism	Immediate	Force
Administrative console	Yes	Yes
JMX `stop` command	Yes	Yes
stopServer command	Yes	No

Note: If the messaging engine reports `isAlive=false` to the HA Manager, the whole application server process is stopped without completions of current transactions or cleanup. This result is equivalent to a forced stop.

Message points

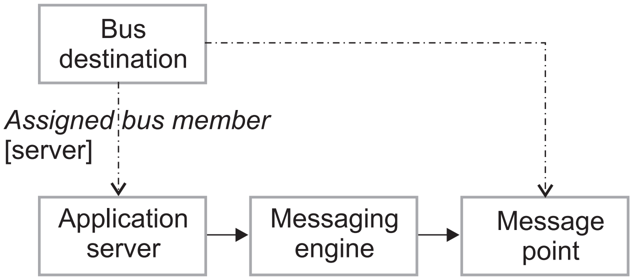
A message point is associated with a messaging engine and holds messages for a bus destination.

A message point is the general term for the location on a messaging engine where messages are held for a bus destination. A message point can be:

- A queue point
- An alias destination
- A publication point
- A mediation point (which is a specialized message point)

For point-to-point messaging, the administrator selects one bus member, which can be an application server, to hold the messages of the queue destination. This action automatically defines a queue point for each messaging engine in the assigned bus member.

- For a queue destination assigned to an application server, all messages sent to that destination are handled by the messaging engine of that server, and message order is preserved.



Applications can use an alias destination to route messages to a target destination in the same bus or to another (foreign) bus (including across a WebSphere MQ link to a queue provided by WebSphere MQ). By assigning an alias destination to a subset of the queue points of a partitioned queue destination, alias destinations can be used to restrict the queue points used by producing and consuming applications.

For publish/subscribe messaging, the administrator configures a topic space destination, but does not have to assign a bus member for the topic space. A topic space has a publication point defined automatically for each messaging engine in the bus.

Message points can be remote from the application which is producing to or consuming from the bus destination. In other words, message points can be on a messaging engine other than the messaging engine to which the application is connected. In this situation the message point is represented at runtime by a *remote message point* on the remote messaging engine.

By monitoring message points and remote message points, you can fully analyze and resolve problems arising from distributed application messaging. For example, you can:

- Determine the state of a specific message request.
- Determine the location of a specific message.
- Examine message queues to determine if messages have been sent or received.
- Free or delete message requests that have become locked.
- Delete or move messages from remote message points.

Remote message points

A remote message point is a messaging engine runtime view of any message point that is associated with a remote messaging engine. Remote message points are dynamically created and destroyed when they are required by the bus; you do not have to configure them explicitly.

Message points provide a physical location to reliably store messages. In a bus that contains many messaging engines, message points can be defined on a subset of the messaging engines in that bus. However, an application can attach to any messaging engine in the bus, and can therefore produce or consume messages to or from destinations that do not have a suitable message point on the messaging engine to which the application is attached.

When an application produces messages, the messages must be moved from the messaging engine for the application to a messaging engine with a suitable message point, and vice versa when an application consumes messages. Remote message points provide a reliable mechanism to move these messages from one messaging engine to another; the remote message points maintain information required to ensure messages are delivered correctly according to the messages' reliability.

Where necessary, messages are queued on a remote message point while awaiting delivery to the intended message point. This runtime information can be monitored and, where appropriate, managed by an administrator.

Each remote message point that exists on a messaging engine has a corresponding representation on the messaging engine that owns the message point.

Message production and consumption by using remote message points:

When an application produces or consumes messages to or from a messaging engine that is not the same as the messaging engine to which the application is connected, remote message points are used to manage the flow of messages between the messaging engines.

Message production

When an application produces messages to a queue-type destination at a messaging engine that is remote from the messaging engine that owns the queue point, a remote queue point is required to manage the delivery of messages destined for the queue point. When an application produces messages to a publish/subscribe type destination, the messaging engine for the producing application will have a local publication point. If subscribing applications to the same destination are attached to different messaging engines in the bus, remote publication points are required to manage the delivery of messages to those remote messaging engines.

If the destination is mediated, messages must first be processed at a mediation point. If the mediation point is on a different messaging engine than the application, a remote mediation point is required to manage the delivery of the messages to the mediation point.

These outbound messages must be delivered to the message point in a reliable manner in accordance with the reliability of the message. To provide these levels of reliability, any message with a reliability greater than “best effort non-persistent” is temporarily queued at the remote message point for the producer messaging engine. The message is queued until the messaging engine that owns the message point confirms the successful arrival of the message, then the producer messaging engine removes its copy of the message from the remote message point. This prevents loss or re-ordering of messages in the event of failures.

Under normal conditions messages will be queued at a remote message point only briefly, but if a failure occurs or the system is overloaded, messages might remain at the remote message point for longer. You can assess the health of the system by monitoring the outbound messages on a remote message point.

Message consumption

A consuming application can be attached to a messaging engine that does not own the store of messages that the application consumes from. When an application consumes from a queue-type destination, the application might be remote from the queue point; when an application consumes from a publish/subscribe type destination, the application might be remote from the subscription. When either of these cases occurs, a remote message point is required to manage message requests made by the application.

Each time an application requests a message from a remote store of messages, a message request is made from the messaging engine for the application to the messaging engine that owns the messages. These message requests are maintained by the remote message point until they are satisfied, either with a message or when the request comes to an end (the requesting application terminates the request).

Point-to-point messaging example by using remote queue points:

When a producing or consuming application is remote from its destination, remote queue points are used to manage the flow of messages between the messaging engine where the destination is located, and the messaging engine to which the application is attached.

The following figure illustrates the use of remote queue points in point-to-point messaging. The producing application attaches to messaging engine ME1, but the bus destination targeted by the application has a queue point on ME2. The queue point on ME2 is represented at runtime by a remote queue point on ME1. The remote queue point receives messages from the application and then reliably transmits them to the queue point on ME2. Likewise, the consuming application attaches to ME3 and consumes messages from the queue point on ME2 through a remote queue point on ME3.

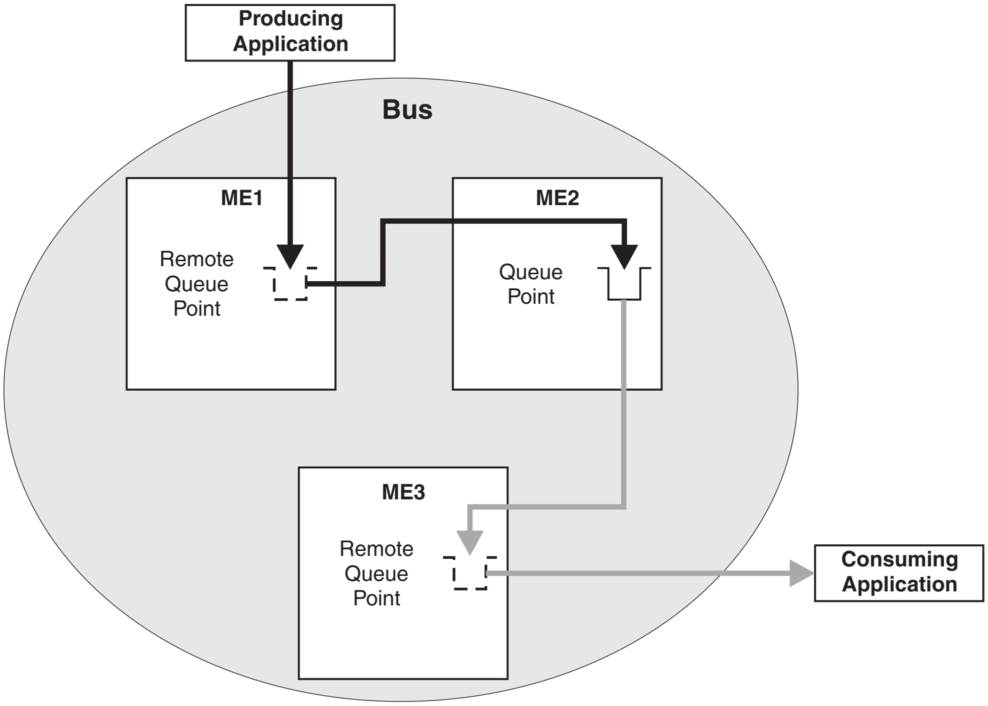


Figure 73. Point-to-point message production and consumption by using remote queue points.

The sequence of steps involved in remote message production is as follows:

1. The producing application, attached to ME1, sends a message to the queue destination, which has a queue point defined on ME2.
2. Messages are queued up on the remote queue point on ME1 before transmission to the queue point on ME2.
3. The message is sent to the queue point on ME2 as soon as possible. ME1 remembers the existence of the message until ME2 confirms that it has received the message.

The sequence of steps involved in remote message consumption is as follows:

1. The consuming application, attached to ME3, attempts to consume a message from the queue destination.
2. ME3 sends a message request to the queue point on ME2.
3. When a message that satisfies the criteria of the message request is available at the queue point on ME2, the message is sent to the remote queue point on ME3.
4. The message is delivered from the remote queue point to the consuming application. If the application consumes the message, the message is deleted from the queue point on ME2. If the application does not consume the message, the message is made available again on the queue point on ME2 for other applications to consume. In either case, the message request is completed and removed from the remote queue point on ME3.

Publish/subscribe messaging example by using remote publication points:

When a publishing or subscribing application is remote from its destination, remote publication points are used to manage the flow of messages between the messaging engine where the destination is located, and the messaging engine the application is attached.

The following diagram illustrates the use of remote publication points in publish/subscribe messaging. Messages are published to a publication point on ME1, and are routed to publication points on ME2 and ME3 through remote publication points on ME1. The messages are consumed from subscriptions on ME2 and ME3.

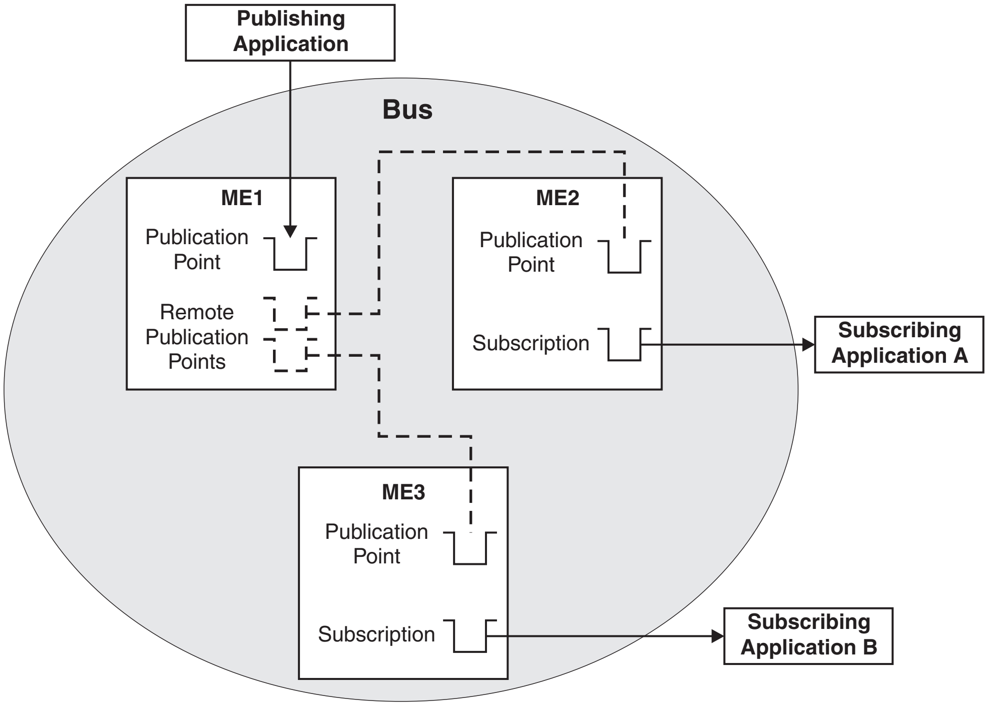


Figure 74. Publish/subscribe messaging by using remote publication points

The sequence of steps involved in remote publish/subscribe messaging is as follows:

1. The administrator creates a topic space destination on the bus; this creates a publication point on each messaging engine in the bus.
2. The subscribing applications register subscriptions for a topic on the topic space on their local messaging engines. ME1 is informed that ME2 and ME3 are interested in the topic.
3. The publishing application, on ME1, publishes a message for that topic and topic space to the bus, for distribution to the publication points on each messaging engine.
4. The remote publication points on ME1 queue the message for transmission to their respective publication points on ME2 and ME3.
5. The message is sent to the publication points on ME2 and ME3 as soon as possible. ME1 remembers the existence of the message until both ME2 and ME3 confirm that they have received the message.
6. The subscribing applications consume the message through their subscriptions on ME2 and ME3.

In figure one, the subscribing applications are attached to the same messaging engines that their subscriptions were created on. If a subscribing application has a durable subscription, it is possible for the application to be attached to a different messaging engine than the messaging engine that the subscription was created on. In this case the subscribing application accesses its subscription through a *remote subscription* on the messaging engine the application is attached. In figure two, messages are published to

ME1, and are routed to the durable subscripton that is on ME2. The messages are consumed from ME2 through a remote subscription on ME3.

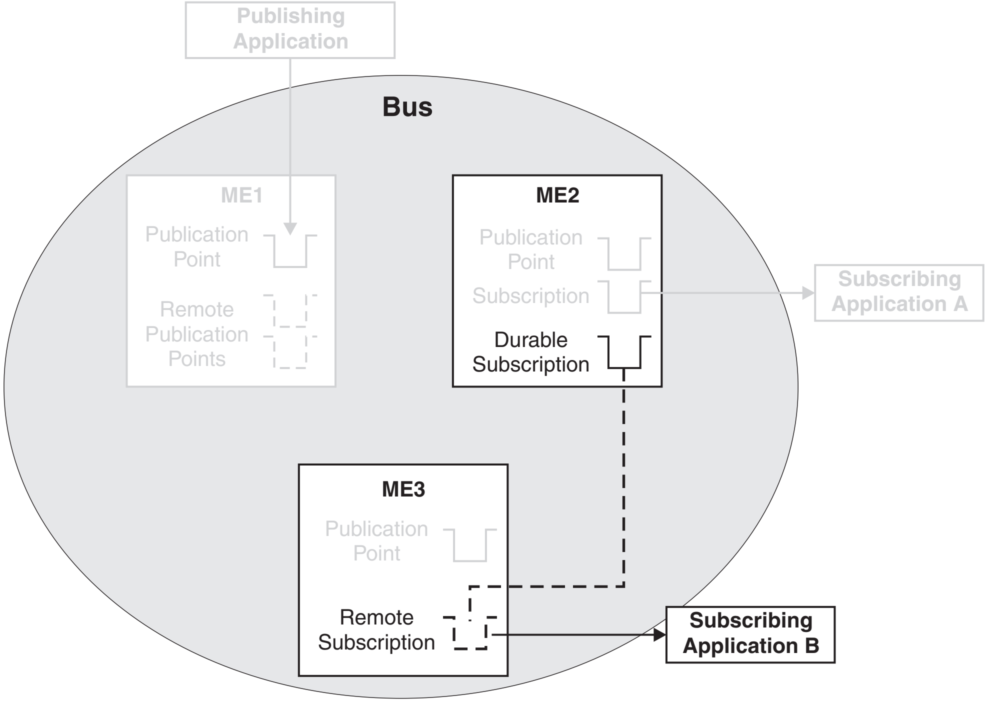


Figure 75. Publish/subscribe messaging using a remote subscription

In this situation, Subscribing Application B consumes messages from its subscription in the same way as an application consumes remotely from a queue point, as detailed in "Point-to-point messaging example by using remote queue points" on page 538.

Messaging engine communication

You can configure different transport options for communication between messaging engines and with WebSphere MQ networks.

The transport channel service provides common networking services, protocols, and I/O operations for the WebSphere Application Server. A channel is a basic functional unit that can be linked with other channels into transport chains. A number of transport chains are defined as part of the application server configuration, and it is also possible to create new transport chains. In this way, the requirements of particular environments are supported.

The transport channel service provides functions for configuring, administering, and initializing chains and their constituent channels. For the purposes of administration, chains are divided into outbound chains and inbound chains. The former are chains used for actively establishing connections, whereas the latter are chains that passively wait for connections to be established.

The inter-operation between service integration and the transport channel service is achieved by implementing two channels:

1. The JFAP channel, which supports intercommunication within service integration
2. The MQFAP channel, which supports WebSphere MQ communication

By using the administrative facilities of the transport channel service, you can assemble these channels into transport chains that support the following protocols:

JFAP over TCP
This is the basic bus protocol; a connection-oriented protocol that uses a standard TCP/IP connection. It includes support for two-phase transactional (remote XA) flows, so that a messaging producer or consumer, running on a client system, can participate in a global transaction managed on that client system.

JFAP over SSL
This is a secure sockets layer (SSL) version of the basic protocol. The protocol starts with the SSL handshake, and if successful continues with the normal data packets, carried using the SSL record format (and encrypted if the SSL handshake established that encryption was required).

JFAP tunneled through HTTP
An HTTP tunnelled version of the basic protocol to enable passing through firewalls.

JFAP tunneled through HTTPS
An HTTPS version of the basic protocol. The protocol starts with an HTTP request, which allows it to be routed by HTTP proxies, and then switches into the SSL variant when the route has been established.

MQFAP over TCP
The basic protocol used for communication with WebSphere MQ.

MQFAP over SSL
A secure sockets layer (SSL) version of the basic protocol used for communication with WebSphere MQ.

You select transports that include SSL or HTTPS, to ensure the security of messages in transit between their producers and consumers. Remember though, that the encryption and decryption of messages by SSL has a high performance and resource cost.

Inbound transport options

There are number of options, such as network and security settings, that apply when configuring messaging engines that receive messages.

The configuration of network transport for service integration is managed through the transport channel service. You can use this service to add, remove, or modify protocols that can be used to establish connections to an application server over a network.

You can configure an application server to allow a combination of several different protocols, that is, a *transport chain*, to be used when communicating with messaging engines hosted by the server. The transport channel service includes support for:

- TCP
- Secure Sockets Layer (SSL) over a TCP network.
- Tunneling through Hyper Text Transfer Protocol (HTTP) connections.
- Tunneling through HTTPS (secure HTTP) connections.

Messaging engine clients such as JMS applications running in a client container and other messaging engines can communicate with a messaging engine using these transport chains.

You can also configure one of two different types of transport chain to be used by WebSphere MQ links and WebSphere MQ client links. These transport chains support:

- TCP
- Secure Sockets Layer (SSL) over a TCP network.

WebSphere MQ queue manager sender channels and WebSphere Application Server applications that use the WebSphere MQ messaging provider can communicate with a messaging engine by using either of these transport chain types.

When a server is created by using the default template, the following transport chains are automatically created to facilitate communication with messaging engines that are hosted by the application server:

InboundBasicMessaging

Allows communication by using the TCP protocol. The default port used by this chain for the first server on the node is 7276. Check the selected port is not already used, for example if you are creating a second server on a particular node. Messaging engines hosted in other application servers and JMS applications running in a client container can communicate with the messaging engines of the server by using this transport chain.

InboundSecureMessaging

Provides secure communication by using the secure sockets layer (SSL) based encryption protocol over a TCP network. The default port used by this chain for the first server on the node is 7286. Check the selected port is not already used, for example if you are creating a second server on a particular node. The SSL configuration information for this chain is based on the default SSL repertoire for the application server. Messaging engines hosted in other application servers and JMS applications running in the client container can communicate using this transport chain.

InboundBasicMQLink

Supports WebSphere MQ queue manager sender channels and applications by using the WebSphere MQ messaging provider connecting over a TCP network. The default port used by this chain is 5558, this can be automatically adjusted to avoid conflicts.

InboundSecureMQLink

Enables WebSphere MQ queue manager sender channels and applications by using the WebSphere MQ messaging provider to establish SSL based encrypted connections over a TCP network. The default port used by this chain is 5578, this is automatically adjusted to avoid conflicts.

soReuseAddr

Allows the WebSphere Application Server administrator to control bind behavior. When the WebSphere Application Server is restarted, if the inbound TCP channels have problems trying to bind the listening socket, errors are printed into the SystemOut file until either the bind is successful or the number of allowed bind attempts has been passed. This custom property helps to avoid repeated error messages during the bind process.

By default all of these transport chains are configured to use the SIBFAPInboundThreadPool thread pool to handle the data they receive. No reason has been identified for it being necessary to change the minimum or maximum size of this thread pool.

You can manage these chains in the administrative console by selecting one of the following:

- **Servers -> Server Types -> WebSphere application servers ->** ***server_name*** **-> [Server messaging] Messaging engine inbound transports**
- **Servers -> Server Types -> WebSphere application servers ->** ***server_name*** **-> [Server messaging] WebSphere MQ link inbound transports**

You can also use these administrative console panels to define new transport chains from a set of templates.

Inbound channel chains that are used for communicating with messaging engines are usually started when the application server that hosts them is started. This can occur even if the application server does not host any active messaging engines. When an inbound chain starts, it binds to the TCP port that it has been assigned and accepts network connections. The following table describes the circumstances under which the inbound chains relating to messaging function are started:

Table 42. Scenarios when inbound chains are started. The first column lists the service integration bus scenarios in which the inbound chains are started. The second column states whether the messaging chains are started for the scenarios. The third column states whether the WebSphere MQ interoperation chains are started for the scenarios.

	Messaging chains	WebSphere MQ interoperation chains
SIB service disabled for server	Not started	Not started
SIB service enabled for server and no WebSphere MQ links or WebSphere MQ client links resources defined	Started	Not started
SIB service enabled and WebSphere MQ links or WebSphere MQ client links resources defined	Started	Started

For more information about enabling or disabling the SIB service, see SIB Service Detail Form.

For more information about defining WebSphere MQ related resources, see, for example, WebSphere MQ link sender channel [Settings].

Note that there is no affinity between a particular inbound channel chain and a messaging engine. Any messaging engine active on a server can be contacted by any inbound channel chain that is running. This has important implications when attempting to secure network communications: communication with the messaging engines that are active in an application server is only as secure as the least secure messaging chain active on the server within the same category, that is, a messaging chain or MQ interop chain.

You can specify inbound transport chains by name in the following places:

- The Inter-engine transport chain field in the Buses [Settings]. This specifies the chain used when establishing connections between nodes in the same cell.
- The Target inbound transport chain field in the Default messaging provider unified connection factory [Settings]. This specifies the transport chain name to use when establishing a network connection for use by a JMS application when connecting to a remote messaging engine.

Outbound transport options

When configuring messaging engines to send messages, configurable options include how they establish connections with other messaging engines or with a WebSphere MQ queue manager that collects and then delivers messages that are received.

The transport channel service manages the configuration of network transports for service integration. However, because manipulation of outbound transport options is an advanced administrative operation, you can carry out some configuration through the **wsadmin** tool.

A number of outbound transport chains are already configured when an application server is created from the default server template, or when a client container is started.

Outbound transport chains are used for either:

- Establishing network connections for the purpose of bootstrapping, which involves establishing a connection with an application server in another cell, for example, because the connection is required for service integration bus links.
- Establishing connections from WebSphere MQ links to WebSphere MQ queue manager receiver channels.

When establishing a network connection, the type of channels and their order in the outbound transport chain used must match those in the inbound transport chain for the server to which connection is made. For example, an outbound HTTP tunneling chain is suitable only for establishing connections with an inbound HTTP tunneling chain.

The following chains are for use during the bootstrap process:

BootstrapBasicMessaging
Used to establish bootstrap connections to inbound chains configured for TCP-only connections to an application server, such as the InboundBasicMessaging chain.

BootstrapSecureMessaging
Used to establish secure connections by using Secure Sockets Layer (SSL) based encryption. The SSL configuration used is taken from the default SSL repertoire when used in an application server environment or from a configuration file when used by the client container. See “Secure transport configuration requirements” on page 546 for more information. This chain can be used for establishing bootstrap connections to inbound chains that are configured to use SSL, for example, the InboundSecureMessaging chain. Success in establishing such a connection depends on a compatible set of SSL credentials being associated with both this bootstrap outbound chain and also with the inbound chain to which the connection is being made.

BootstrapTunneledMessaging
Used to connect when tunneling through HTTP.

BootstrapTunneledSecureMessaging
Used to establish bootstrapping connections that are tunneled through secure HTTP (HTTPS). Like the BootstrapSecureMessaging outbound chain, this chain also derives its SSL configuration from the default SSL repertoire when used in an application server or from a configuration file when used in the client container. See “Secure transport configuration requirements” on page 546.

The outbound chains that an application server uses for bootstrap operations are defined when the server is defined. They can be altered, or new bootstrap chains can be defined, by using the **wsadmin** tool. See Defining outbound chains for bootstrapping and Defining outbound chains for WebSphere MQ interoperation.

You cannot configure bootstrap outbound chains used by a client container. However you can configure some attributes of outbound chains that use SSL encryption protocols. For more information, see “Secure transport configuration requirements” on page 546.

You need the names of outbound bootstrap chains when configuring:

- The Provider endpoints of a JMS connection factory; see Default messaging provider unified connection factory [Settings].
- The Bootstrap endpoints of a service integration bus link; see Service integration bus links [Settings].

The following chains can be used when establishing a network connection to a WebSphere MQ queue manager receiver channel:

OutboundBasicMQLink
Used to establish connections with WebSphere MQ queue manager receiver channels.

OutboundSecureMQLink
Used to establish connections with WebSphere MQ queue manager receiver channels that have been secured using SSL. The SSL configuration used is taken from the default SSL repertoire for the application server being used to contact the queue manager.

The names of outbound chains used for WebSphere MQ interoperation are needed when configuring the transport chain of a WebSphere MQ link.

By default all of these transport chains are configured to use the SIBFAPThreadPool thread pool to send data. No reason has been identified for it being necessary to change the minimum or maximum size of this thread pool.

Secure transport configuration requirements

There are additional configuration requirements when configuring secure transport, such as inbound chains, to establish SSL-based or HTTPS-based connections between messaging engines, or between messaging engines and JMS applications running in a client container.

For an SSL connection to be established successfully, the party that is initiating the connection and the party that is waiting for the connection to be made must both supply a compatible set of credentials.

When you are configuring the client container to bootstrap using an SSL-based transport chain, you specify additional SSL properties in the *sib.client.ssl.properties* properties file. This file is located in the *profile_root/properties* directory of the application server installation, where *profile_root* is the directory in which profile-specific information is stored. The properties in this file are used for all client container bootstrapping activities over both SSL and HTTPS-based bootstrap chains.

You can override or augment properties specified in the sib.client.ssl.properties file by specifying system properties of the same name to the application client. Do this by specifying a -CCD command line option naming the property and its new value. For more information about command line syntax, see launchClient tool.

Note: Some of the properties in the *sib.client.ssl.properties* file duplicate those in the *sas.client.props* file. Overriding these properties by using wsadmin command options affects both sets of properties.

When you are configuring SSL-based connections between two messaging engines, both the messaging engines must have inbound chains with matching names. These inbound chains must be configured with compatible sets of SSL credentials. The compatibility must be true for both intra-bus messaging engine connections and for connections between messaging engines that are in different buses.

A particular inbound transport chain must have no affinity with a messaging engine. Any enabled inbound transport chain can contact any messaging engine that is active on a server because by default, an application server is created with unsecured inbound transport chains. Disable or delete these chains to restrict access to secure chains only.

Security for messaging engines

When bus security is enabled, you need to be aware of the additional requirements to secure communication between messaging engines.

To ensure that messaging engines operate securely when bus security is enabled, you should understand the following points:

- Use secure transport connections (SSL or HTTPS) to ensure confidentiality and integrity of messages in transit between messaging engines. Define an appropriate secure transport chain, and then reference the transport chain name from the bus propertyInter-engine transport chain. For more information, see "Secure transport configuration requirements."
- If the bus has a bus member at WebSphere Application Server Version 6, set the Inter-engine authentication alias property. This prevents unauthorized clients or messaging engines from establishing a connection. For more information, see Adding a secured bus.
- Secure access to the data store for a messaging engine by using a user ID and password. Apply higher levels of security by using the underlying features of message stores. For example, for a data store, Apache Derby Version 10.3 allows the whole database to be encrypted, DB2 allows specific tables to be encrypted. These features must be managed directly by the appropriate database administrator. Refer to Securing database access for more details.
- If fine-grained administrative security is in use, messaging engines are administered as resources at the server or cluster level.

Applications with a dependency on messaging engine availability

If an application depends on a messaging engine being available, then the messaging engine must be started before the application can be run.

If you want the application server to start an application automatically, you must develop your application to test that any required messaging engine has been started and, if needed, wait for the messaging engine to start. If this is technique used in a startup bean, then the startup bean method should perform the work (to test and wait) in a separate thread, and use the standard WorkManager methods, so that the application server startup is not delayed.

For an example of code to test and wait for a messaging engine, see the following code extract:

```
import java.util.Iterator;
import javax.management.ObjectName;
import com.ibm.websphere.management.AdminService;
import com.ibm.websphere.management.AdminServiceFactory;

    String messagingEngineName = "messagingEngineName";
    // Messaging engine to check if started? for example "node01.server1-bus1"
    boolean meStarted = false;

    AdminService adminService = AdminServiceFactory.getAdminService();
    while (!meStarted) {
      String filterString = "WebSphere:type=SIBMessagingEngine,name=" +
                            messagingEngineName + ",*";
      boolean foundBean = false;
      ObjectName objectName = null;
      try {
        ObjectName objectNameFilter = new ObjectName(filterString);
        Iterator iter = adminService.queryNames(objectNameFilter,null).iterator();
        while (iter.hasNext()) {
            objectName = (ObjectName) iter.next();
            foundBean = true;
            break;
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
      if (foundBean) {
        // You have found messaging engine MBean, which means it is initialized,
        // now check if it is in Started state?
        meStarted =
          ((Boolean) adminService.invoke(objectName, "isStarted", null, null)).booleanValue();
      }

      if (!meStarted) {
        // messaging engine is not started yet so sleep (wait) for a bit...
        Thread.sleep(5000);
      }
    }
```

Bus destinations

Service integration has the following types of bus destinations each with a different purpose: queue, topic space, foreign, and alias.

You can create and administer the following types of service integration bus destination:

Queue destination
A queue destination represents a message queue and is used for point-to-point messaging. A service integration queue destination is localized in a particular bus member (application server or cluster of application servers). When a producer sends a message to the queue destination, the service integration bus delivers the message to a messaging engine in that bus member. The

messaging engine then delivers the message to a consumer. If necessary, the messaging engine queues the message until a consumer is ready to receive it.

Topic space destination

A topic space destination represents a set of “publish and subscribe” topics and is used for publish/subscribe messaging. The topic for a specific message (publication) is a property of the message.

A service integration topic space destination is not localized in a particular bus member. Service integration maintains a list of subscriptions in the topic space and matches each publication against that list. When a new publication matches one or more subscriptions in the topic space, service integration delivers a copy of the publication to each subscriber. If necessary, service integration can queue the publication message until the subscriber is ready to receive it. If the new publication does not match any subscription, service integration discards the publication.

Foreign destination

A foreign destination represents a destination that is defined in another bus (a foreign bus). You can use a foreign destination for point-to-point messaging. You use a foreign destination if you need to override security settings, or messaging defaults, for an individual destination on a foreign bus.

The foreign bus can be another service integration bus or a WebSphere MQ network (that is, one or more interconnected WebSphere MQ queue managers or queue-sharing groups). When a producer sends a message to a foreign destination, service integration delivers the message to the foreign bus. The foreign bus is then responsible for queueing the message, as appropriate, based on its definition of the destination.

To override messaging defaults of a destination on a foreign bus, you configure the properties (for example, the default priority) of the foreign destination. To override security settings and control which users and groups in the local bus have access to a destination in a foreign bus, you configure the destination roles of the foreign destination. These properties apply when an application that is connected to the local bus sends messages to the destination in the foreign bus.

You do not use foreign destinations for publish-subscribe messaging. Instead, applications publish messages locally by using a topic space destination in the local bus, and you configure a service integration bus link or a WebSphere MQ link. These links propagate the published messages into the foreign bus, or buses, where subscribers receive the messages. For a link to a service integration bus, configure topic space mappings, as described in Configuring topic space mappings between service integration buses. For a link to a WebSphere MQ network, configure a publish/subscribe bridge, as described in “Publish/subscribe messaging through a WebSphere MQ link” on page 421.

Alias destination

An alias destination maps an alternative name for a bus destination. You can use an alias destination for point-to-point messaging or publish/subscribe messaging. An alias destination maps a bus name and destination name (identifier) to a target where the bus name, or the destination name, or both, are different. An alias destination can map to a queue destination or a topic space destination. If required, alias destinations can be chained so that the target destination is itself an alias destination.

You use an alias destination when you need to make a destination available under an alternative name. For example:

- Service integration destinations might have names that do not comply with WebSphere MQ naming restrictions (for example, the names are too long). For such destinations, you can define an alias destination that maps a WebSphere MQ-compliant name to the service integration name. A WebSphere MQ application can use the WebSphere MQ-compliant name to send messages to the destination.

- You can assign an alias destination to a subset of the queue points of a partitioned queue destination, and therefore use the alias destination to restrict the queue points that the producing and consuming applications use.

When you use an alias destination, you can also set properties (for example, the default quality of service) for the alias destination. When an application uses the alias destination, these properties override the properties of the target destination. If you do not want to override a property, configure the alias destination to inherit the corresponding property from the target destination.

When you use an alias destination, you can also configure destination roles for the alias destination. When the application uses the alias destination, service integration in the local bus uses these roles to control which users and groups in the local bus have access to the target destination. If you do not want to override the security for the target destination, configure the alias destination to delegate the authorization check to the target destination.

Bus destinations can be either permanent or temporary. When an administrator configures a service integration destination, that destination is a permanent destination that exists until an administrator explicitly deletes it. In contrast, a temporary destination exists only while an application is using it. Typically, this situation occurs when the application uses a JMS temporary destination. Service integration creates a corresponding temporary service integration bus destination.

You can configure queue, topic space, and alias destinations with one or more *mediations*. Mediations are programs that process each message after the producing application sends the message to the destination, and before any consuming applications receive the message from the destination. For example, a mediation can modify the actual message, or redirect the message to another destination or sequence of destinations, or both.

You can configure queue, topic space, and alias destinations with *routing paths*:

- The default forward routing path defines a sequential list of intermediary destinations that messages must pass through to reach the target destination, before consumers can retrieve the messages from that destination. Each intermediary destination applies its mediations to the messages.
- The reply destination is the next destination to which reply messages are sent.
- “How JMS destinations relate to service integration destinations”
- “Queue destinations” on page 551
- “Publish/subscribe messaging and topic spaces” on page 552
- “Foreign destinations and alias destinations” on page 555
- “Permanent bus destinations” on page 559
- “Temporary bus destinations” on page 560
- “Exception destinations” on page 560
- “Destination mediation” on page 562
- “Destination routing paths” on page 562
- “Message points” on page 536
- “Message ordering” on page 569
- “Strict message ordering for bus destinations” on page 571
- “Message selection and filtering” on page 573

How JMS destinations relate to service integration destinations

Most WebSphere Application Server applications use the JMS APIs to access the services provided by the service integration bus. JMS defines JMS destinations, which are the Java objects to which JMS applications send messages and from which JMS applications receive messages. The attributes of a JMS destination include the address of the destination that the messaging provider uses. For the service integration messaging provider, this address is a service integration destination name (a queue name or topic space name) and a bus name. In this way, a JMS destination can identify a service integration bus destination.

Typically, a JMS application obtains a JMS destination from JNDI lookup of the destination JNDI name. However, a JMS application can also obtain a JMS destination in other ways, for example, from the JMSReplyTo property of a JMS message.

JMS destinations - queues and topics

A JMS destination can be one of the following destination types:

JMS queue destination

Used for point-to-point messaging, in which producing applications (producers) send messages to a queue. The messaging provider stores just one copy of each message until a consuming application (consumer) receives the message. If there are several consumers, only one consumer receives a copy of the message; if there are no consumers, the message is queued.

In service integration, a JMS queue destination object has a queue name property and a bus name property (it also has other properties).

JMS topic destination

Used for publish/subscribe messaging, in which producing applications (publishers) send messages (publications) to a topic. The messaging provider delivers a copy of each publication to each consuming application (subscriber). If there are no subscribers, service integration discards the publication.

Another difference from point-to-point messaging is that subscribers can consume messages from multiple similar topics by including wildcards in a topic name (publishers cannot include wildcards in a topic name).

In service integration, a JMS topic destination object has a topic name, a topic space name, and a bus name property (it also has other properties).

JMS destinations - relationship with service integration destinations

In service integration, a JMS destination identifies a service integration destination. Its queue name or topic space name property is the name of the service integration destination. Its bus name property is the name of the service integration bus that contains the destination.

You can omit the bus name property when you define the JMS destination. If you do then the JMS destination identifies the service integration destination in the local bus; that is, whichever bus the JMS application connects to. This can be convenient where there is only one service integration bus or where all buses contain a destination with the same name.

Service integration includes the following destination types:

Service integration queue destination

A queue destination represents a message queue and is used for point-to-point messaging. A service integration queue destination is localized in a particular bus member (application server or cluster of application servers). When a producer sends a message to the queue destination, the service integration bus delivers the message to a messaging engine in that bus member. The messaging engine then delivers the message to a consumer. If necessary, the messaging engine queues the message until a consumer is ready to receive it.

Typically, a JMS queue destination identifies a service integration queue destination; that is, its bus name property matches the local bus name and its queue name property matches the name of a service integration queue destination in the local bus.

Service integration topic space destination

A topic space destination represents a set of “publish and subscribe” topics and is used for publish/subscribe messaging. The topic for a specific message (publication) is a property of the message. A service integration topic space destination is not localized in a particular bus member. Service integration maintains a list of subscriptions in the topic space and matches each publication against that list. When a new publication matches one or more subscriptions in the topic space, service integration delivers a copy of the publication to each subscriber. If necessary, service integration can queue the publication message until the subscriber is ready to receive it. If the new publication does not match any subscription, service integration discards the publication.

Typically, a JMS topic destination identifies a service integration topic space destination; that is, its bus name property matches the local bus name and its topic space name property matches the name of a service integration topic space destination in the local bus. When a JMS application sends a message to the JMS topic destination, service integration sets the destination topic property of the message to the topic name property of the JMS topic destination and then sends the message to the service integration topic space destination.

Service integration foreign destination

A foreign destination represents a destination that is defined in another bus (a foreign bus). You can use a foreign destination for point-to-point messaging. You use a foreign destination if you need to override security settings, or messaging defaults, for an individual destination on a foreign bus. The foreign bus can be another service integration bus or a WebSphere MQ network (that is, one or more interconnected WebSphere MQ queue managers or queue-sharing groups). When a producer sends a message to a foreign destination, service integration delivers the message to the foreign bus. The foreign bus is then responsible for queueing the message, as appropriate, based on its definition of the destination.

A JMS destination can identify a service integration foreign destination; that is, its bus name and queue or topic space name properties can match the foreign bus name and queue or topic space name of the foreign destination. However, this is not always necessary. If there is no service integration foreign destination with a matching foreign bus name and a matching destination (queue or topic space) name, service integration sends the message to the specified foreign bus anyway.

Queue destinations

A queue destination represents a message queue and is used for point-to-point messaging. A service integration queue destination is localized in a particular bus member (application server or cluster of application servers). When a producer sends a message to the queue destination, the service integration bus delivers the message to a messaging engine in that bus member. The messaging engine then delivers the message to a consumer. If necessary, the messaging engine queues the message until a consumer is ready to receive it..

The term “queue” is used, as an abbreviation for “queue destination”, to refer to a bus destination configured for point-to-point messaging.

Note: Applications use API-specific artifacts such as a JMS queue, which encapsulates the name of a queue destination, but are unaware of the existence of the service integration destination or of the bus it is configured on. For more information, see “How JMS destinations relate to service integration destinations” on page 549.

The administrator assigns a queue destination to only one member (an application server or server cluster) of the service integration bus, or a WebSphere MQ server. The messaging engine in the bus member hosts the message point for the queue, known as a queue point. The queue point is the location where messages for the queue are stored and processed.

If the bus member has more than one messaging engine, the queue destination is partitioned across the messaging engines. Each messaging engine has a separate queue point. Each message that is sent to a queue destination is held on only one of the destination queue points: the message is not duplicated across queue points. Each messaging engine handles a share of the messages arriving at the destination so that the messaging workload is balanced across all messaging engines in the cluster bus member.

An application can also create its own temporary queues, which appear temporarily in the list of queue points for the messaging engine, but usually need no administrative intervention.

When a destination is configured on a bus member that is a WebSphere MQ server, the destination has a single queue point called a WebSphere MQ queue point. This WebSphere MQ queue point is a WebSphere MQ queue on the WebSphere MQ queue manager or queue-sharing group that the WebSphere MQ Server represents. For more information about WebSphere MQ queue points, see related concepts.

Publish/subscribe messaging and topic spaces

You can use publish/subscribe messaging to publish one message to many subscribers. A producing application publishes a message on a given subject area or topic. The topic for a specific message (publication) is a property of the message. Consumer applications that have subscribed to the topic each receive a copy of the message. A topic space is a hierarchy of publish/subscribe topics. These topics have publication points automatically defined on each messaging engine in their associated service integration bus.

A service integration topic space destination is not localized in a particular bus member. Service integration maintains a list of subscriptions in the topic space and matches each publication against that list. When a new publication matches one or more subscriptions in the topic space, service integration delivers a copy of the publication to each subscriber. If necessary, service integration can queue the publication message until the subscriber is ready to receive it. If the new publication does not match any subscription, service integration discards the publication.

A topic space has a set of default publish/subscribe permissions for all topics in the hierarchy. An administrator can configure individual publish/subscribe topics with specific permissions and mediations. Publish/subscribe topics inherit such configurations from higher publish/subscribe topics in the topic space hierarchy and the topic space itself.

Publish/subscribe topics with the same name can exist in multiple topic spaces, but there can be only one topic space with a given name in the bus. For example, consider a publish/subscribe topic hierarchy split into the following topic spaces:

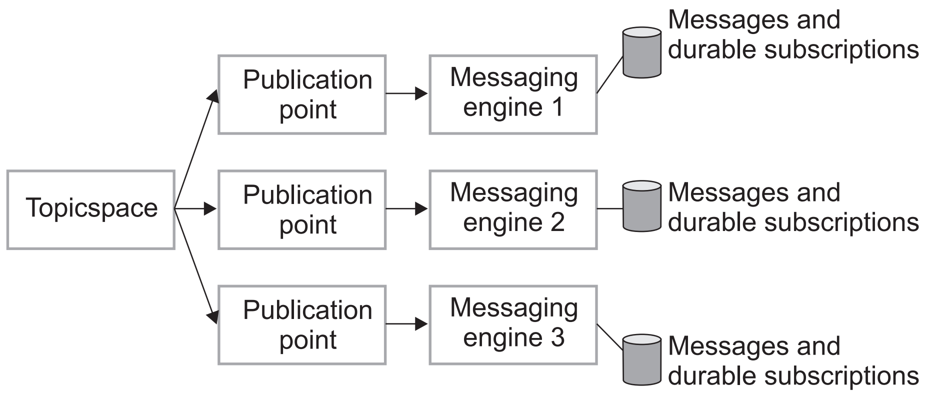
`library`
: Topics for document management.

`sales`
: Topics for marketing and sales tracking.

The publish/subscribe topic “`volumes`” can appear in the topic hierarchy in both topic spaces, and can have different meanings in each.

A producing application can attach to any messaging engine on the bus. Messages are stored in the message store for the messaging engine to which the producer is attached. As a result, a topic space can have messages stored in a number of message stores at various (producing) messaging engines in the bus. Messages for a publish/subscribe topic are published to a publication point and automatically forwarded to all other publication points for which there are subscribers on that topic.

Figure 76. Publish/subscribe messaging



The default messaging provider supports durable subscriptions to publish/subscribe topics. These enable a subscriber to receive a copy of all messages published to a topic, even messages published during periods of time when the subscriber is not connected to the server. For a given JMS connection factory or activation specification, all publish/subscribe messages to be delivered to durable subscriptions are stored on the publication point of the messaging engine named by the durable subscription home property. Therefore if that messaging engine is unavailable, subscribers cannot retrieve messages. These undelivered messages are preserved and sent to the durable subscriptions after the messaging engine restarts.

The consequences of changing durable subscriptions

When an application connects to an existing durable subscription, but specifies parameters that differ from those that were used to create the existing subscription, the subscription is deleted then recreated with the new parameters. A durable subscription can be changed in this way only when it has no active consumers.

In the basic case, there is only one active consumer at any time, so the application can change the durable subscription without affecting other subscribers. However, the situation is more complicated for cloned subscriptions.

A cloned durable subscription has multiple active subscribers, which are usually clones of a particular application.

- For cloned message-driven bean (MDB) applications, the subscribers are always active on the subscription, and so the administrator must stop all instances of the MDB application before the subscription can be altered. (If the MDB application instances are recycled one at a time then each individual instance is thrown out when it tries to connect using the changed properties, because there are existing consumers.)
- For cloned EJB applications, administrators should ensure that all instances of the EJB application are stopped before the subscription can be changed, to avoid the following problem. Enterprise beans have active subscribers for a durable subscription for relatively short spaces of time. If the EJB application instances are recycled individually then there is a period in which different instances of the application have different views of the subscription configuration. This causes the subscription to be deleted and recreated which can lead to the loss of messages. Subscriptions can be deleted and recreated multiple times until the new definition is constant across all instances.

Note: The `*server_name*-durableSubscriptions.ser` file in the `*WAS_HOME*/temp` directory is used by the messaging service to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription. If you have to delete the `*WAS_HOME*/temp` directory or other files in it, ensure that you preserve this file.

Topic names and use of wildcard characters in topic expressions

Wildcard characters can be used in topic expressions to retrieve topics provided by the default messaging provider and service integration technologies.

Each subscribe request includes a topic expression that identifies one or more topics that the subscription is to be associated with, and that the request uses to match against incoming messages.

Subscription topic expressions for the default messaging provider and service integration technologies are based on a subset of the XPath location path syntax.

Identifying individual topics

Every topic in a topic space has a *topic name* consisting of one or more name parts, separated by / (forward slash) characters:

```
Topic name = name_part | (name_part '/' topic_name)
```

Using wildcards to identify multiple topics

To select one or more topics in a topic space, you can use a *topic path*, a location path that contains wildcard characters. Topic spaces are evaluated by using a subset of the XPath location path syntax with the <topicspace> element as the initial context node, so that non-wildcarded topic paths look exactly like topic names.

The syntax for topic paths can be summarized as follows:

- A topic path that contains no * (asterisk), // (double forward slash), or . (dot) symbols is asking for an exact match with the topic name specified.
- A * (asterisk) can be used as a wild card and matches one level (regardless of the value of the name part at that level)

  A * can be used anywhere in a topic path expression, but if it isn't at the start it must be preceded by a /, and if it is isn't at the end it must be followed by a /
- // can be used as a wild card and matches 0 or more levels

  A // can be used anywhere in the expression except at the end. To match 0 or more levels at the end of the expression you end the expression with the syntax //. (double-slash dot). To match one or more levels at the end use //* (double-slash asterisk)

  A topic path must not contain more than two consecutive / symbols.

The following table lists some example topic paths showing the XPath syntax and the equivalent WBI Message Broker selectors:

Table 43. XPath syntax and WBI Message Broker selectors. The first column of the table lists some topic path examples. The second column displays the topics selected in the path. The third column provides the equivalent WBI Message Broker selectors.

Topic path	Topics selected	WBI Message Broker equivalent
A/B	Selects the B child of A	A/B
A/*	Selects all children of A	A/+
A//*	Selects all descendents of A	A/#/+
A//.	Selects A and all descendents of A	A/#
//*	Selects everything	# (or #/+)
A/./B	Equivalent to A/B	A/B
A/*/B	Selects all B grandchildren of A	A/+/B
A//B	Selects all B descendents of A	A/#/B

Table 43. XPath syntax and WBI Message Broker selectors (continued). The first column of the table lists some topic path examples. The second column displays the topics selected in the path. The third column provides the equivalent WBI Message Broker selectors.

Topic path	Topics selected	WBI Message Broker equivalent
//A	Selects all A elements at any level	#/A
*	Selects all first level elements	+

Foreign destinations and alias destinations

Foreign destinations and alias destinations are types of bus destination. A foreign destination represents a destination that is defined in another bus (a foreign bus). An alias destination maps to an alternative name for a bus destination that is defined either in the local bus or in a foreign bus.

Usually, you do not need to configure a foreign destination or an alias destination:

- For an application to send messages to a destination that is defined in the local bus, you specify the bus name and the destination name in the JMS destination object (queue or topic).

  It is possible to omit the bus name, because the default is the local bus name, but for a system with more than one bus, it is advisable to specify the bus name.
- For an application to send messages to a destination that is defined in a foreign bus, you specify the bus name (that is, the foreign bus) and the destination name in the JMS destination object (queue or topic). You do not need to configure any destination objects in the local bus.

  Service integration uses the definition of the foreign bus that is configured on the local bus. This definition includes default values for the destination attributes, such as the default quality of service. These default values apply to all destinations in that foreign bus. For more information, see the topic about point-to-point messaging across multiple buses.

You use a foreign destination when you need to override messaging defaults, security settings, or both for an individual destination on a foreign bus. You define a foreign destination on the local bus. When an application that is connected to the local bus sends messages to the destination in the foreign bus, the attributes of the foreign destination override the destination default values. You can set properties and destination roles, but you cannot map to an alternative name for the destination.

You use an alias destination when you need to use an alternative name for a bus destination. The bus destination can be on the local bus or on a foreign bus. You configure an alias destination on the local bus. When an application in the local bus uses the alias destination, the specified bus name and destination name are mapped to a new name. If you use an alias destination, you can also set properties, destination roles, or both.

When an application that is connected to a bus specifies a destination name and bus name in its JMS destination object (queue or topic) that match the identifier and bus of an alias destination that is defined in that bus, the destination that the application accesses is the same as if the application specified the target identifier and target bus from the alias destination. You can also use a alias destination that is defined in a foreign bus if you need to redirect messages that arrive over a foreign bus connection to differently named destinations or buses, and you cannot modify the configuration of the source bus.

Foreign destinations

A foreign destination represents a destination that is defined in another bus (a foreign bus). You can use a foreign destination for point-to-point messaging. You use a foreign destination if you need to override security settings, or messaging defaults, for an individual destination on a foreign bus. The foreign bus can be another service integration bus or a WebSphere MQ network (that is, one or more interconnected WebSphere MQ queue managers or queue-sharing groups).

To override messaging defaults of a destination on a foreign bus, you configure the properties (for example, the default priority) of the foreign destination. To override security settings and control which users and groups in the local bus have access to a destination in a foreign bus, you configure the destination roles of the foreign destination. These properties apply when an application that is connected to the local bus sends messages to the destination in the foreign bus.

When you define a foreign destination, use the actual names of the foreign bus and the destination on the foreign bus, so that the JMS destination object does not change.

When an application that is connected to the local bus sends messages to the destination in the foreign bus, service integration in the local bus uses the properties and destination roles of the foreign destination, rather than the default values from the definition of the foreign bus (on the local bus). Typically, you configure the properties of a foreign destination to match the properties that are configured for that destination in the foreign bus (where that destination is a local destination), but this is not essential.

You can also configure destination roles for the foreign destination. Service integration in the local bus uses these roles to control which users and groups in the local bus have access to the destination. It also complements any access controls that the foreign bus applies.

You do not use foreign destinations for publish-subscribe messaging. Instead, applications publish messages locally using a topic space destination in the local bus, and you configure a service integration bus link or a WebSphere MQ link. These links propagate the published messages into the foreign bus, or buses, where subscribers receive the messages. For a link to a service integration bus, configure topic space mappings, as described in Configuring topic space mappings between service integration buses. For a link to a WebSphere MQ network, configure a publish/subscribe bridge, as described in “Publish/subscribe messaging through a WebSphere MQ link” on page 421.

Service integration cannot use configuration information that is scoped to a foreign bus. Therefore, if an appropriate foreign destination is not defined on the local bus, service integration uses default values for the destination attributes.

Figure 1 shows a JMS application that sends messages from the local bus, Bus 1, to a destination in a foreign bus, Bus 2. Bus 1 has a foreign bus connection defined, which it uses to forward the message to the foreign bus. The foreign destination is not defined in the local bus. Bus 1 gets the destination defaults from the foreign bus connection.

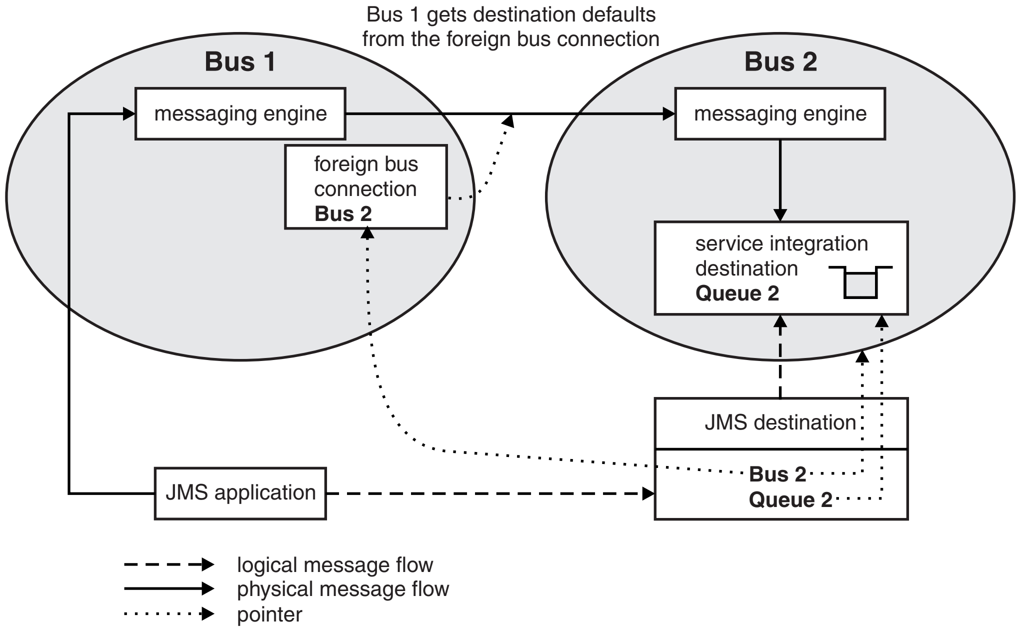


Figure 77. Point-to-point messaging between two buses with no foreign destination configured

Figure 2 shows a JMS application that sends messages from the local bus, Bus 1, to a destination in a foreign bus, Bus 2. Bus 1 has a foreign bus connection defined, which it uses to forward the message to the foreign bus. Bus 1 includes a foreign destination definition. Bus 1 gets the destination defaults from the foreign destination.

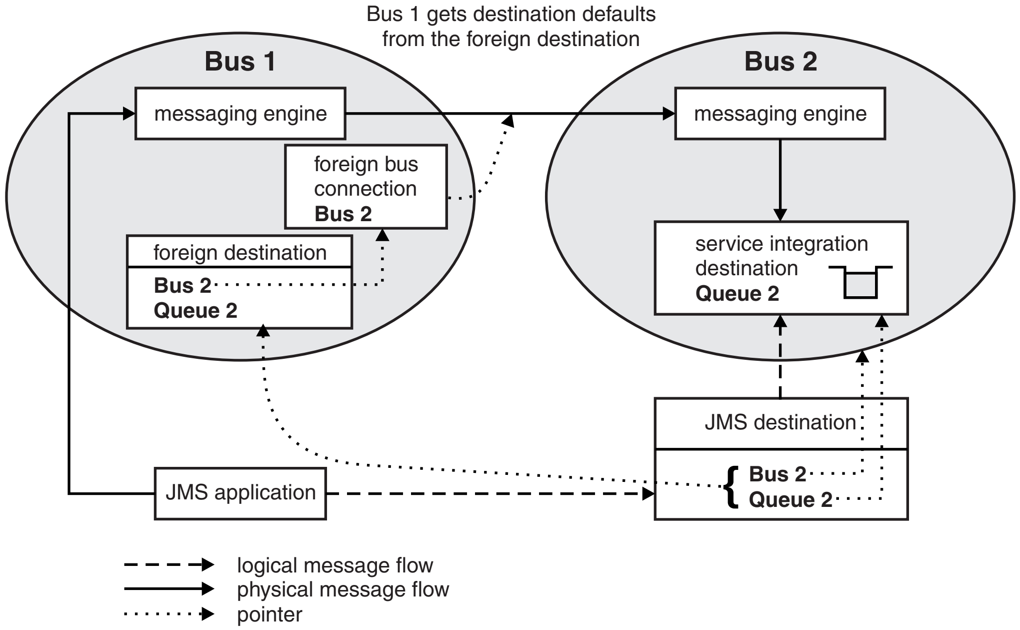


Figure 78. Point-to-point messaging between two buses with a foreign destination configured on the local bus

Alias destinations

An alias destination maps an alternative name for a bus destination. You can use an alias destination for point-to-point messaging or publish/subscribe messaging. An alias destination maps a bus name and destination name (identifier) to a target where the bus name, or the destination name, or both, are different. An alias destination can map to a queue destination or a topic space destination. If required, alias destinations can be chained so that the target destination is itself an alias destination.

You use an alias destination when you need to make a destination available under an alternative name. For example:

- You need to interoperate with WebSphere MQ, but some service integration bus names or destination names do not comply with WebSphere MQ naming restrictions (for example, the names are too long). You can define an alias destination that maps a WebSphere MQ-compliant name to the service integration name.

  For example, an application sends a message to a WebSphere MQ application and the reply-to destination name does not comply with the WebSphere MQ naming restrictions. You can define an alias that maps a compliant name to the actual reply-to destination name. The application then specifies the alias destination as the reply-to.

  Another example is an application that sends a message through a WebSphere MQ foreign bus to a remote service integration foreign bus when the send-to destination name does not comply with the WebSphere MQ naming restrictions. You can define an alias in the remote bus that maps a compliant name to the actual send-to destination name. The application then specifies the alias destination as the send-to. If you want the sending application to use the actual destination name, you can define an alias in the local bus that maps the actual destination name to the compliant name.
- If you move a destination from one bus to another (by deleting it, then creating it on another bus), you can create an alias destination that redirects messages from the old destination to the new one. You must create the alias destination in every bus where applications reference the destination, for example, the bus that the destination is moved from, and the bus that the destination is moved to.

  However, it might be simpler to change the JMS destinations that are registered with JNDI to point to the new destination.
- You can assign an alias destination to a subset of the queue points of a partitioned queue destination, and therefore use the alias destination to restrict the queue points that the producing and consuming applications use.

When you use an alias destination, you can also set properties (for example, the default quality of service) for the alias destination. When an application uses the alias destination, these properties override the properties of the target destination. If you do not want to override a property, configure the alias destination to inherit the corresponding property from the target destination.

When you use an alias destination, you can also configure destination roles for the alias destination. When the application uses the alias destination, service integration in the local bus uses these roles to control which users and groups in the local bus have access to the target destination. If you do not want to override the security for the target destination, configure the alias destination to delegate the authorization check to the target destination.

Figure 3 shows a JMS application that sends messages from the local bus, Bus 1, to a destination in a foreign bus, Bus 2. Bus 1 has a foreign bus connection defined, which it uses to forward the message to the foreign bus. The JMS destination does not point to the target queue, but points to Bus X, Queue Y. Bus 1 includes an alias destination that maps Bus X, Queue Y to the target destination Bus 2, Queue 2. Bus 1 gets the destination defaults from the alias destination.

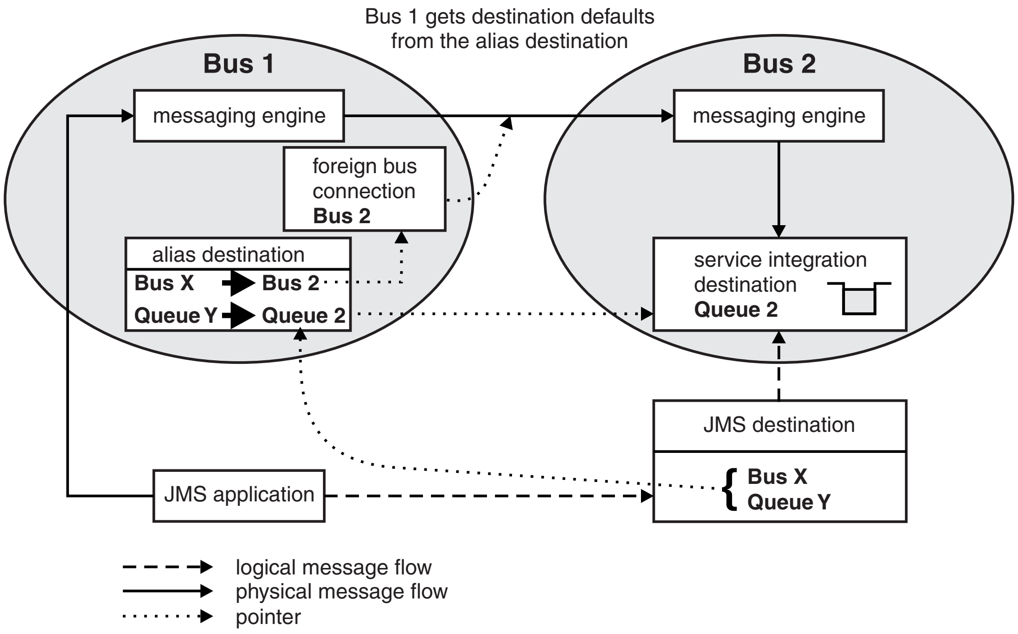


Figure 79. Point-to-point messaging between two buses with an alias destination configured on the local bus

Using an alias destination for a destination in a foreign bus

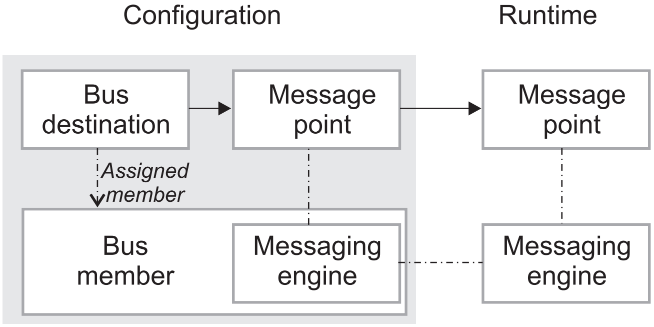
If an application uses an alias destination to access a destination that is defined in a foreign bus, you can configure the alias destination with the specific properties, destination roles, or both that the application requires. This means that you probably do not need to define a foreign destination as well.

Permanent bus destinations

A permanent destination is configured by an administrator and has its runtime instances created automatically by the messaging provider. Permanent destinations remain until the administrator explicitly deletes them.

When you configure a destination on a bus member that is a WebSphere MQ server, the destination has a single queue point called a *WebSphere MQ queue point*. This *WebSphere MQ queue point* is a WebSphere MQ queue on the WebSphere MQ queue manager or queue-sharing group that the WebSphere MQ Server represents. For more information about WebSphere MQ queue points, see related concepts.

The configuration and runtime components of a permanent destination are shown in the following figure.



The administrator configures a *bus destination*, to define properties such as the type of destination, and mediations and quality of service for the destination. A bus destination is defined on a service integration bus, and is hosted by one or more bus members, depending on whether it is a queue destination or a topic space destination. This results in the system generating one or more *message points*, where messages are held on that bus. A message point can be configured to override some properties inherited from the bus destination.

For point-to-point messaging, the administrator configures the destination as a queue and selects one bus member, an application server, as the assigned bus member that is to hold messages for the queue. This action automatically defines a queue point (a type of message point) for each messaging engine in the assigned member. When an assigned messaging engine starts up, a runtime instance of each of its message points is automatically created. You can use the runtime view of a message point to administer the messages on that location.

For publish/subscribe messaging, the administrator configures the destination as a topic space (a hierarchy of topics), but does not have to select any assigned bus member for the topic space. A topic space has a publication point (a type of message point) defined automatically for each messaging engine in the bus.

Temporary bus destinations

A temporary destination only exists while an application is using it. Typically, this situation occurs when the application uses a JMS temporary destination. Service integration creates a corresponding temporary service integration bus destination.

For example, if an application creates a temporary JMS queue for use with the default messaging provider, the SIB service automatically creates a temporary queue destination on the bus.

A temporary destination is assigned a unique name specific to the SIB service. The name starts with the prefix `_Q` for temporary queues or `_T` for temporary topics.

Temporary destinations appear on the list of runtime queue and publication points for a messaging engine on the service integration bus. Temporary destinations do not usually need administration.

Temporary topics cannot be explicitly created on the service integration bus. API-specific temporary destinations (for example, JMS temporary topics) are implemented by creating a temporary topic space and using a single topic within the temporary topic space. This is not apparent to the application code. For this reason, applications cannot use a wildcard for temporary topics. This means, for example, a subscriber cannot use a wildcard to receive a copy of messages published to all temporary topics.

See also “How JMS destinations relate to service integration destinations” on page 549.

Exception destinations

An exception destination is a location for messages that cannot be delivered to, or remain on, a specified target destination, but that also cannot be discarded. Exception destinations prevent the loss of messages when this is required by the quality of service specified for a message.

An exception destination can be used in the following situations:

- Service integration cannot deliver a message to the specified target destination, and cannot discard the message because of the quality of service of the message. Service integration delivers the message to an exception destination.
- A message exceeds the maximum number of delivery attempts to a transactional consumer. This situation might occur if the transactional consumer fails and the message backs out and is consumed again repeatedly. When the delivery limit (the Maximum failed deliveries per message) is reached, the message goes to the exception destination.

- A destination that has messages on it is deleted. Those messages are moved to an exception destination.

For each of these situations, you can configure which exception destination processing to use:

- Use the default exception destination of the relevant messaging engine.

  Each messaging engine has a default exception destination called `_SYSTEM.Exception.Destination.`*`messaging_engine_name`* that is created automatically when a messaging engine is created. This default exception destination stores messages that cannot be delivered for bus destinations that are localized to the messaging engine. When you use the default exception destination, an administrator can access all messages that cannot be delivered for a messaging engine in a single location.

  Note: You cannot modify the default exception destination and you must not delete it.
- Use a specific exception destination that is associated with the relevant resource, for example, a queue destination, a topic space destination, a service integration bus link or a WebSphere MQ link.

  The exception destination that is associated with a destination is used if a message cannot be delivered because the number of delivery attempts to a transactional consumer is exceeded. When you use a specific exception destination for a queue or topic space destination, an administrator can access those undeliverable messages for that destination in one location.

  The exception destination that is associated with a link is used if a message cannot be delivered because the target destination is full or does not exist.

  An exception destination must be a queue destination, and can be local or remote. The exception destination must already exist before you configure another resource to use that exception destination. If the exception destination is not a queue, or if it does not exist when the message arrives, messages are rerouted to the default exception destination of the relevant messaging engine.

  Note that you cannot configure an exception destination for a bus; you must configure an exception destination for each destination on the bus.
- Do not reroute undeliverable messages to any exception destination, that is, specify `None`.

  Attempts to deliver the message continue. For a service integration bus link, an undeliverable message might block the processing of other messages waiting for delivery to the same destination. For a WebSphere MQ link, an undeliverable message might block the processing of other messages waiting for delivery through that link to the same bus.

The report options that are set in the properties of individual messages can affect exception destination processing. Depending on the report option that is set, when the conditions apply for service integration to send a message to an exception destination, service integration also sends a report message to the reply-to destination of the message, or discards the message instead of sending it to the exception destination, or both.

Note:

- Best-effort messages are always discarded if they cannot be delivered to their target destination, that is, they never use an exception destination.
- A message cannot be available to consumers until it is successfully delivered to a destination.

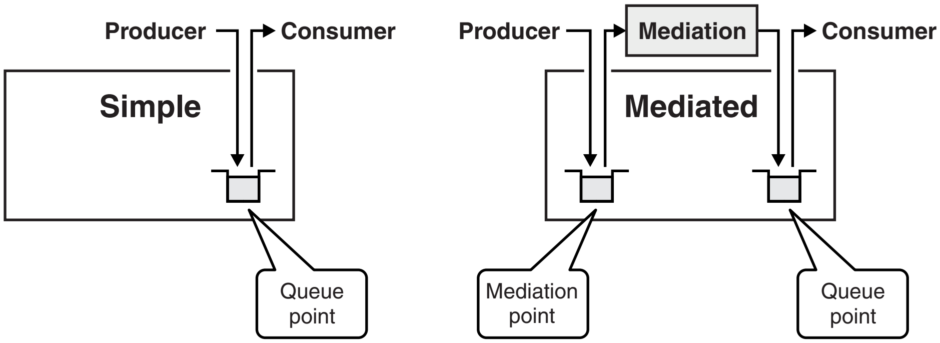
Service integration cannot guarantee the ordering of messages sent to an exception destination. Because of this, if message order is important, you can configure a bus destination so that it does not use an exception destination. In this situation, the Maximum failed deliveries per message limit specified for the destination is ignored, and the message remains available to consumers. Synchronous consumers repeatedly attempt to get the message; message-driven beans and other asynchronous consumers repeatedly attempt consume the message. This situation continues until either the message is removed from the destination (for example, by an administrator using the administrative console) or the consumer can subsequently process the message without rolling back.

Destination mediation

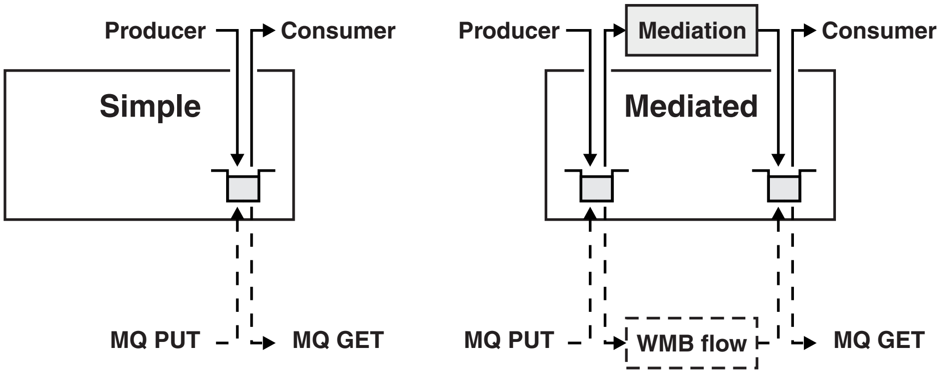
A destination can be configured with one or more mediations that refine how messages are handled by the destination. For example, a mediation can modify the actual message, or redirect the message to another destination or sequence of destinations, or both.

A mediation can process messages through message transformation, subsetting, aggregation, disaggregation, and using a selection of destinations (but not consumers) to which the message can be forwarded.

When a message arrives at the mediation point, the mediation consumes the message and can transform, subset, aggregate or disaggregate the message. The message is then either forwarded to another destination or returned to the same destination. If the message is returned to the same destination, it goes to the queue point, where it can be consumed by the messaging application. This process is shown in the following figure:



You can configure a destination so the mediation point or the queue point, or both are WebSphere MQ queues. If both are WebSphere MQ queues then a WebSphere MQ application can act as an external mediation as shown in the following figure:



Destination routing paths

A routing path defines a sequential list of intermediary bus destinations that messages must pass through to reach a target bus destination. A routing path is used to apply the mediations configured on several destinations to messages sent along the path.

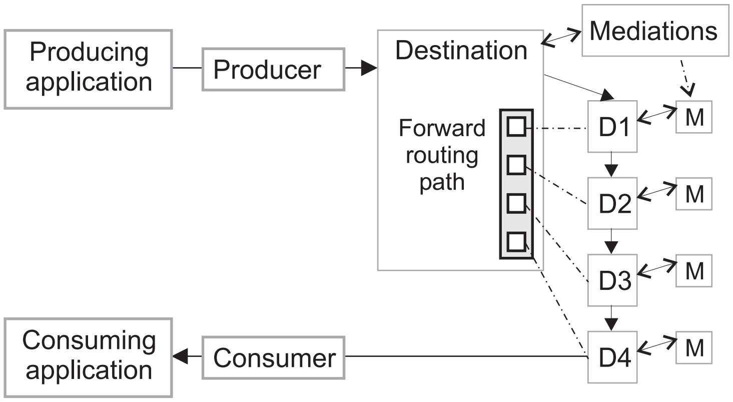


Figure 80. Routing paths

A *forward routing path* identifies a list of bus destinations that a message should be sent to, from the producer to the last destination from which consumers retrieve messages. The *reverse routing path* is constructed automatically for request/reply messages, and identifies the list of destinations that any reply message should be sent to, from the consumer back to the producer. Use of reverse routing path enables a reply message to take a different route back to the producer, and therefore have more mediations applied.

When a message arrives at a destination in the path, mediations can manipulate the entries in the forward routing path, to change the sequence of destinations through which messages pass. If a mediation manipulates the forward routing path, and the reverse routing path is set for a request message that expects a reply, the mediation is responsible for making any corresponding changes to the reverse routing path.

A destination without mediations can be included in a routing path to provide a future option to apply a mediation assigned to that destination.

Do not include a topic space in a forward routing path.

If the first element of the routing path does not represent a destination known to the bus, the message is sent to the exception destination defined for the current destination.

Forward routing paths

A producer can attach to one destination and pass messages along a *forward routing path* to the target destination that consumers use.

- The producer can set the forward routing path in the original message.
- An administrator can configure a default forward routing path on destinations for use by messages that do not contain a forward routing path.

When a message is sent to a destination (either directly, or by following its forward routing path), and prior to invoking any mediation at the destination, the `Default forward routing path` property of the destination is applied to the forward routing path of the message, as follows:

- If the incoming message contains an empty forward routing path, the forward routing path in the message is set to the value of the `Default forward routing path` property of the destination (which also can be empty or null).
- If the incoming message has a non-empty forward routing path, it is left unchanged. Once you have applied any mediations, the message is forwarded to the next destination in the path. When the last destination in the path is reached, the message is handled by that destination.

Reverse routing paths

A producer can ask for reply messages by specifying a reply destination in messages it sends. The *reverse routing path* is constructed dynamically as the message passes from one destination to another along the forward routing path.

When a message is sent to a destination (either directly, or by following its forward routing path), and prior to invoking any mediation at the destination, the `Reply destination` property of the destination is applied to any non-empty reverse routing path of the message, as follows:

- If the incoming message has a non-empty reverse routing path (indicating that a reply is expected), the value of the `Reply destination` property of the destination is added to front of the reverse routing path in the message (indicating that the reply message must visit this new reply destination before any destinations that are already in the reverse routing path).
- If the incoming message has an empty reverse routing path (indicating that a reply is not expected), the reverse routing path is left unchanged.

Message points

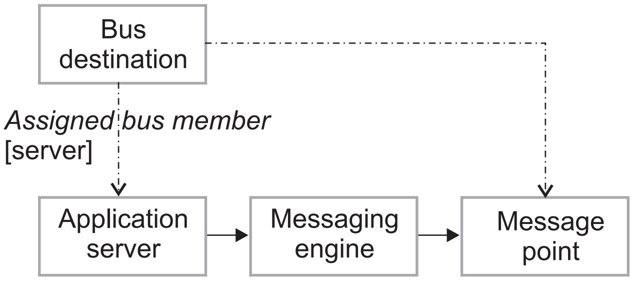
A message point is associated with a messaging engine and holds messages for a bus destination.

A message point is the general term for the location on a messaging engine where messages are held for a bus destination. A message point can be:

- A queue point
- An alias destination
- A publication point
- A mediation point (which is a specialized message point)

For point-to-point messaging, the administrator selects one bus member, which can be an application server, to hold the messages of the queue destination. This action automatically defines a queue point for each messaging engine in the assigned bus member.

- For a queue destination assigned to an application server, all messages sent to that destination are handled by the messaging engine of that server, and message order is preserved.



Applications can use an alias destination to route messages to a target destination in the same bus or to another (foreign) bus (including across a WebSphere MQ link to a queue provided by WebSphere MQ). By assigning an alias destination to a subset of the queue points of a partitioned queue destination, alias destinations can be used to restrict the queue points used by producing and consuming applications.

For publish/subscribe messaging, the administrator configures a topic space destination, but does not have to assign a bus member for the topic space. A topic space has a publication point defined automatically for each messaging engine in the bus.

Message points can be remote from the application which is producing to or consuming from the bus destination. In other words, message points can be on a messaging engine other than the messaging engine to which the application is connected. In this situation the message point is represented at runtime by a *remote message point* on the remote messaging engine.

By monitoring message points and remote message points, you can fully analyze and resolve problems arising from distributed application messaging. For example, you can:

- Determine the state of a specific message request.
- Determine the location of a specific message.
- Examine message queues to determine if messages have been sent or received.
- Free or delete message requests that have become locked.
- Delete or move messages from remote message points.

Remote message points

A remote message point is a messaging engine runtime view of any message point that is associated with a remote messaging engine. Remote message points are dynamically created and destroyed when they are required by the bus; you do not have to configure them explicitly.

Message points provide a physical location to reliably store messages. In a bus that contains many messaging engines, message points can be defined on a subset of the messaging engines in that bus. However, an application can attach to any messaging engine in the bus, and can therefore produce or consume messages to or from destinations that do not have a suitable message point on the messaging engine to which the application is attached.

When an application produces messages, the messages must be moved from the messaging engine for the application to a messaging engine with a suitable message point, and vice versa when an application consumes messages. Remote message points provide a reliable mechanism to move these messages from one messaging engine to another; the remote message points maintain information required to ensure messages are delivered correctly according to the messages' reliability.

Where necessary, messages are queued on a remote message point while awaiting delivery to the intended message point. This runtime information can be monitored and, where appropriate, managed by an administrator.

Each remote message point that exists on a messaging engine has a corresponding representation on the messaging engine that owns the message point.

Message production and consumption by using remote message points:

When an application produces or consumes messages to or from a messaging engine that is not the same as the messaging engine to which the application is connected, remote message points are used to manage the flow of messages between the messaging engines.

Message production

When an application produces messages to a queue-type destination at a messaging engine that is remote from the messaging engine that owns the queue point, a remote queue point is required to manage the delivery of messages destined for the queue point. When an application produces messages to a publish/subscribe type destination, the messaging engine for the producing application will have a local publication point. If subscribing applications to the same destination are attached to different messaging engines in the bus, remote publication points are required to manage the delivery of messages to those remote messaging engines.

If the destination is mediated, messages must first be processed at a mediation point. If the mediation point is on a different messaging engine than the application, a remote mediation point is required to manage the delivery of the messages to the mediation point.

These outbound messages must be delivered to the message point in a reliable manner in accordance with the reliability of the message. To provide these levels of reliability, any message with a reliability greater than “best effort non-persistent” is temporarily queued at the remote message point for the producer messaging engine. The message is queued until the messaging engine that owns the message point confirms the successful arrival of the message, then the producer messaging engine removes its copy of the message from the remote message point. This prevents loss or re-ordering of messages in the event of failures.

Under normal conditions messages will be queued at a remote message point only briefly, but if a failure occurs or the system is overloaded, messages might remain at the remote message point for longer. You can assess the health of the system by monitoring the outbound messages on a remote message point.

Message consumption

A consuming application can be attached to a messaging engine that does not own the store of messages that the application consumes from. When an application consumes from a queue-type destination, the application might be remote from the queue point; when an application consumes from a publish/subscribe type destination, the application might be remote from the subscription. When either of these cases occurs, a remote message point is required to manage message requests made by the application.

Each time an application requests a message from a remote store of messages, a message request is made from the messaging engine for the application to the messaging engine that owns the messages. These message requests are maintained by the remote message point until they are satisfied, either with a message or when the request comes to an end (the requesting application terminates the request).

Point-to-point messaging example by using remote queue points:

When a producing or consuming application is remote from its destination, remote queue points are used to manage the flow of messages between the messaging engine where the destination is located, and the messaging engine to which the application is attached.

The following figure illustrates the use of remote queue points in point-to-point messaging. The producing application attaches to messaging engine ME1, but the bus destination targeted by the application has a queue point on ME2. The queue point on ME2 is represented at runtime by a remote queue point on ME1. The remote queue point receives messages from the application and then reliably transmits them to the queue point on ME2. Likewise, the consuming application attaches to ME3 and consumes messages from the queue point on ME2 through a remote queue point on ME3.

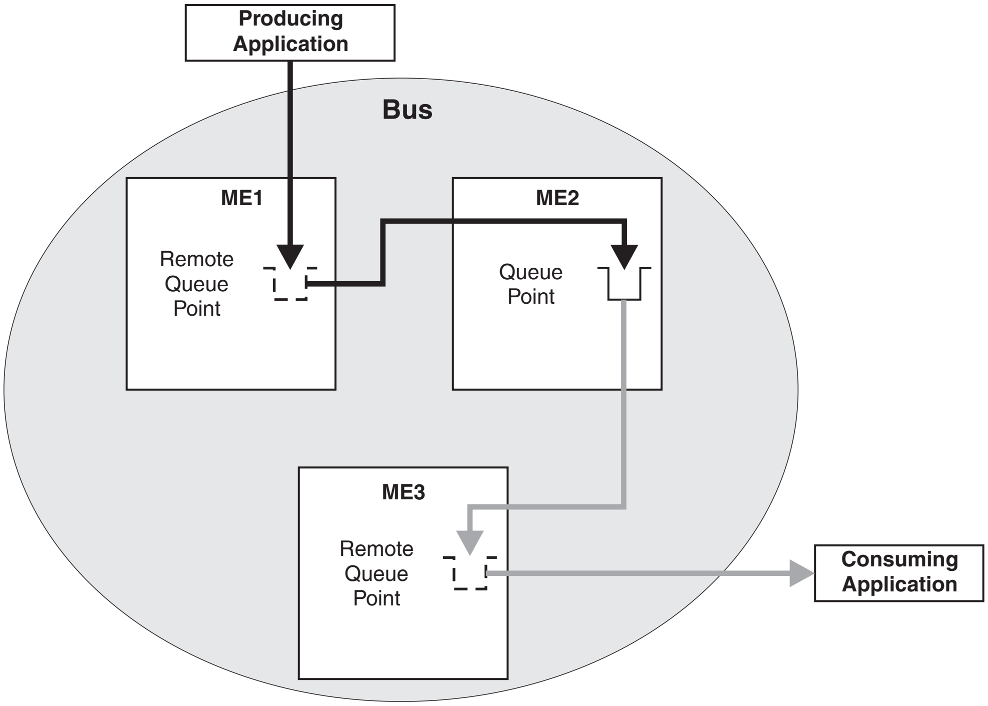


Figure 81. Point-to-point message production and consumption by using remote queue points.

The sequence of steps involved in remote message production is as follows:

1. The producing application, attached to ME1, sends a message to the queue destination, which has a queue point defined on ME2.
2. Messages are queued up on the remote queue point on ME1 before transmission to the queue point on ME2.
3. The message is sent to the queue point on ME2 as soon as possible. ME1 remembers the existence of the message until ME2 confirms that it has received the message.

The sequence of steps involved in remote message consumption is as follows:

1. The consuming application, attached to ME3, attempts to consume a message from the queue destination.
2. ME3 sends a message request to the queue point on ME2.
3. When a message that satisfies the criteria of the message request is available at the queue point on ME2, the message is sent to the remote queue point on ME3.
4. The message is delivered from the remote queue point to the consuming application. If the application consumes the message, the message is deleted from the queue point on ME2. If the application does not consume the message, the message is made available again on the queue point on ME2 for other applications to consume. In either case, the message request is completed and removed from the remote queue point on ME3.

Publish/subscribe messaging example by using remote publication points:

When a publishing or subscribing application is remote from its destination, remote publication points are used to manage the flow of messages between the messaging engine where the destination is located, and the messaging engine the application is attached.

The following diagram illustrates the use of remote publication points in publish/subscribe messaging. Messages are published to a publication point on ME1, and are routed to publication points on ME2 and

ME3 through remote publication points on ME1. The messages are consumed from subscriptions on ME2 and ME3.

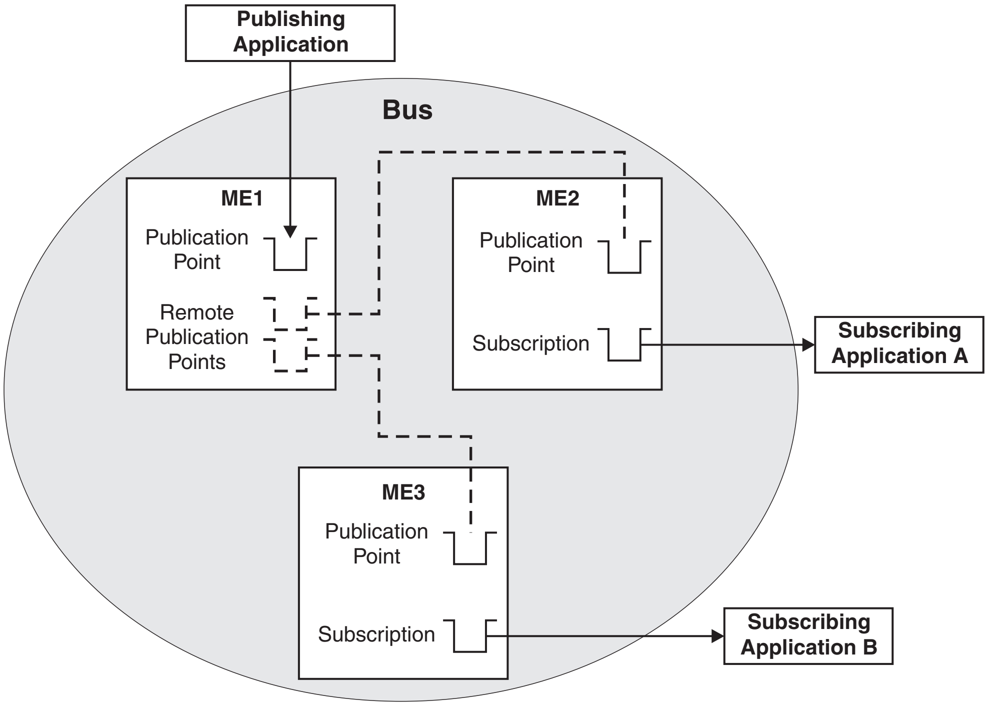


Figure 82. Publish/subscribe messaging by using remote publication points

The sequence of steps involved in remote publish/subscribe messaging is as follows:

1. The administrator creates a topic space destination on the bus; this creates a publication point on each messaging engine in the bus.
2. The subscribing applications register subscriptions for a topic on the topic space on their local messaging engines. ME1 is informed that ME2 and ME3 are interested in the topic.
3. The publishing application, on ME1, publishes a message for that topic and topic space to the bus, for distribution to the publication points on each messaging engine.
4. The remote publication points on ME1 queue the message for transmission to their respective publication points on ME2 and ME3.
5. The message is sent to the publication points on ME2 and ME3 as soon as possible. ME1 remembers the existence of the message until both ME2 and ME3 confirm that they have received the message.
6. The subscribing applications consume the message through their subscriptions on ME2 and ME3.

In figure one, the subscribing applications are attached to the same messaging engines that their subscriptions were created on. If a subscribing application has a durable subscription, it is possible for the application to be attached to a different messaging engine than the messaging engine that the subscription was created on. In this case the subscribing application accesses its subscription through a *remote subscription* on the messaging engine the application is attached. In figure two, messages are published to ME1, and are routed to the durable subscripton that is on ME2. The messages are consumed from ME2 through a remote subscription on ME3.

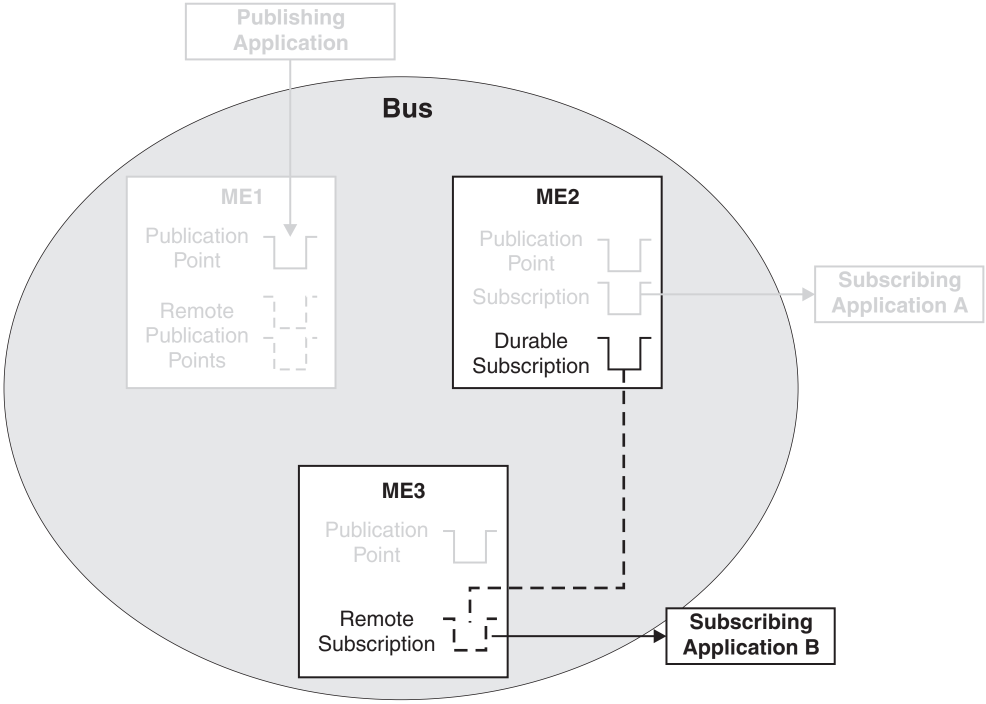


Figure 83. Publish/subscribe messaging using a remote subscription

In this situation, Subscribing Application B consumes messages from its subscription in the same way as an application consumes remotely from a queue point, as detailed in “Point-to-point messaging example by using remote queue points” on page 538.

Message ordering

Message ordering is important to some asynchronous messaging applications; that is, it is important to consume messages in the same order that the producer sends them. If this type of message ordering is important to your application, your design must take it into account.

For example, a messaging application that processes seat reservations might have producer components and a consumer component. A producer component sends a message to the consumer component when a customer reserves a seat. If the customer cancels the reservation then the producer (or possibly a different producer) sends a second message. Typically, the consumer component must process the first message (which reserves the seat) before it processes the second message (which cancels the reservation).

Some applications use a synchronous (request-response) pattern where the producer waits for a response to each message before it sends the next message. In this type of application, the consumer controls the order in which it receives the messages and can ensure that this is the same order as the producer or producers send them. Other applications use an asynchronous (fire and forget) pattern where the producer sends messages without waiting for responses. Even for this type of application, order is usually preserved; that is, a consumer can expect to receive messages in the same order as the producer or producers send them, especially when there is a significant time between sending consecutive messages. However your design must consider factors that can disrupt this order.

The order of messages is disrupted if your application sends messages with different priorities (higher priority messages can overtake lower priority messages) or if your application explicitly receives a message other than the first by specifying message selectors. Parallel processing and error or exception processing can also affect message ordering.

Parallel processing

Multiple destinations

When a producer sends a message to one destination and then sends a second message to a different destination, it is possible that the second message arrives at its destination before the first message arrives at its destination. This can happen even if the producer sends both messages within the same transaction (the transaction ensures that either both sends fail or both complete; it does not ensure the delivery order). If this is a problem for your application then avoid using more than one destination for that application.

Multiple producers

When one producer sends a message to a destination and then a second producer sends a second message to the same destination, it is possible that the second message arrives at the destination before the first message. This can happen even if the second producer synchronizes with the first producer, for example by waiting for a signal from the first producer before sending its message (the second message). It can also happen even if the producers are transactional (the fact that a transaction is complete does not mean that the message or messages are at the destination - service integration can complete a transaction first and deliver the message or messages later). If this is a problem for your application then avoid using more than one producer for that application. Similarly, avoid using a single producer which runs multiple parallel threads.

Multiple consumers

When two messages arrive at a queue-type destination where there are multiple consumers, it is possible that one consumer receives the first message and another receives the second. If the consumers can run in parallel then the consumer which receives the second message might process it before the consumer which receives the first message. Also, if one consumer receives a message transactionally and then rolls-back the transaction, the message returns to the destination where another consumer can receive it; meanwhile, that other consumer might already have received and processed other (later) messages. If either of these are a problem for your application then avoid using more than one consumer for that application. Similarly, do not allow multiple concurrent message-driven bean (MDB) invocations per endpoint (see Configuring MDB or SCA throttling for the default messaging provider). Alternatively, consider using strict message ordering, see “Strict message ordering for bus destinations” on page 571.

Publish and subscribe

For publish-subscribe messaging, service integration does not guarantee the ordering of messages received by different instances of a subscription, including separate instances of a durable subscription created with the Share Durable Subscriptions property set to permit sharing.

Errors and exception conditions

Quality of service

The quality of service for a message determines how error and exception conditions can affect the delivery of the message (see “Message reliability levels - JMS delivery mode and service integration quality of service” on page 635. Depending on the quality of service, an error or exception condition can cause service integration to deliver the message reliably (exactly once) or less reliably (more than once, or not at all). For example, with the express nonpersistent quality of service, it is possible for sequences of messages to be re-delivered after a failure; however, their underlying order is always maintained so that messages might arrive in the sequence: 1, 2, 3, 2, 3, 4. More generally, service integration does not guarantee the ordering of different messages sent with different qualities of service. If this is a problem for your application then be sure to select an appropriate quality of service and avoid mixing different qualities of service for messages to the same destination.

Exception destinations

Under certain failure scenarios, service integration can deliver a message to an exception destination or deliver a message to a destination and subsequently transfer the message to an exception destination. Service integration does not guarantee the ordering of messages it sends to or transfers to an exception destination. If this is a problem for your application then you can configure the destination so that it does not use an exception destination (see “Exception destinations” on page 560). If the producer and the destination are in different buses then you can configure the link or links between those buses to have no exception destination (see “Foreign buses” on page 618 and Configuring exception destination processing for a link to a foreign bus.

Transaction roll-back

The activation specification for a message-driven bean (MDB) can specify a delay between failing message retries. This delay allows time for the condition which causes the roll-back to recover before the same message drives the MDB again. However, while a message is delayed in this way, another (later) message can drive the MDB (see Protecting an MDB or SCA application from system resource problems). If this is a problem for your application then consider limiting the maximum sequential failures to one (see JMS activation specification [Settings] or consider using strict message ordering, see “Strict message ordering for bus destinations.”

Resolving indoubt transactions

During two-phase commit processing, an application can fail while a JMS send or receive operation is in an indoubt state. When this happens, it is possible for the application to restart before the indoubt state is resolved so that one or more messages are "invisible" when the application restarts but become "visible" later. Other messages on the destination are not affected. A consuming application can receive a message (say message 2) which is logically after an "invisible" message (say message 1); later, after the indoubt transaction is resolved, the application receives the previously "invisible" message (message 1). In this way, the application might receive and process messages in the wrong order. If this is a problem for your application then consider using strict message ordering, see “Strict message ordering for bus destinations.”

Strict message ordering for bus destinations

If message order must be maintained in all circumstances, a destination can be configured so that the order of messages is preserved much more rigorously than for a normal destination.

In general, messages produced by a single producer to a single destination are seen by a consumer to that destination in the same order as they are produced. However, there are certain topologies and events, such as system failures, that can change the order of messages (see “Message ordering” on page 569 for details). If strict message order is essential then you can configure a destination to preserve the order of the messages in a much more rigorous manner. Using strict message ordering, in conjunction with restricted topologies (as described later in this section), maintains message order in all circumstances.

Restrictions that are enforced automatically

When you include a strictly ordered destination in your system, certain restrictions are automatically enforced at run time. These restrictions affect the way that applications can interact with the destination to ensure strict message order, ensure that you fully understand each of the following restrictions prior to configuring a destination to be totally ordered.

1. Concurrent consumers are prevented from attaching to an ordered destination. Having more than one consumer consuming from a destination can result in messages being consumed out of order. So, for an ordered queue destination, and any subscription on an ordered topic space, one consumer at most can attach at any one time. This is equivalent to setting “receive exclusive” to `true` on a queue destination and “Share durable subscriptions” to “never shared” on a JMS topic connection factory, and warnings are generated in the system log to indicate if these options have been overridden. The attached consumer is also restricted to a single transaction at any one time (this is the normal behavior for many messaging interfaces so should not affect many users, for example, JMS).

2. Partially consumed messages prevent subsequent messages from being consumed. For standard destinations (that is, destinations without the strict message ordering option enabled), messages that have been partially consumed from the destination (for example, received within a transaction that is yet to be committed) might be skipped over by a consumer to allow processing of messages to continue past previously consumed messages that are yet to be committed. This can disrupt message order. For an ordered destination, such messages are not skipped by a consumer, instead the consumer is blocked until the message has either been totally removed (for example, the uncommitted transaction is committed) or replaced (for example, the uncommitted transaction is rolled back). This situation would occur if a previously attached consumer fails to commit a transaction that was used to consume messages from the destination, or if a transactional resource becomes temporarily unavailable during the commit of a transaction. If the resource is permanently unavailable, see Resolving indoubt transactions for information on resolving these transactions.
3. Concurrent message driven beans (MDBs) are restricted for an ordered destination. The maximum concurrent endpoints and maximum batch size settings of any MDB deployed on an ordered destination are overridden to be one. For more information, see JMS activation specification [Settings]. This ensures the ordered processing of messages by the MDB. When the system overrides this property at run time, a warning is generated in the system log.
4. Concurrent mediations are restricted for an ordered destination. The allow concurrent mediation setting of any mediation on an ordered destination is overridden to be false. For more information, see “Concurrent mediations” on page 603. This ensures the ordered processing of messages by the mediation. When the system overrides this property at run time, a warning is generated in the system log.
5. If **`Automatically stop endpoints on repeated message failure`** is enabled then the **`Sequential failed message threshold`** is overridden to 1.

Additional restrictions that might affect message ordering

Even with strict message order enabled on a destination the following restrictions are not automatically enforced, but they could affect message ordering and therefore should be understood:

- If a destination has an exception destination configured, it is possible for messages intended for that destination to be delivered to the exception destination under error conditions such as a message reaching its maximum failed delivery limit. This redirecting of messages could be regarded as changing the order of messages, and if that is so, then the destination should set the exception destination to “none”. Note that setting the exception destination to none results in messages that would otherwise be moved to the exception destination remaining on the destination.
- Changing the topology of a service integration bus can affect message order. For example, deleting and recreating an ordered destination must be seen as two distinct destinations and therefore order cannot be guaranteed across both destinations.
- Introducing or removing a mediation can affect message order. If a mediation is introduced to a destination or is removed from a destination while messages are flowing to and through the destination, message order is not necessarily preserved.
- Mediations can be designed to deliberately affect message order, and therefore changing a mediation can directly affect message order. For example, you could introduce a mediation that reorders messages or routes messages to different destinations.
- Alias destinations and foreign destinations do not have an option to maintain strict message order. In each case, only the underlying destination can be ordered.
- If a queue destination is deployed to a cluster bus member that has more than one messaging engine, which means that the destination has more than one queue point or mediation point, message order cannot be maintained across the destination. Therefore, enabling strict message order on such a destination does not assure message order. Configuring message affinity allows sets of messages to be sent to the same queue point for processing in order by a single consumer. Message affinity has performance implications because the messages are no longer workload balanced across multiple queue points.

Important: The system does not prevent you from using partitioned destinations. It is your responsibility to ensure that partitioned destinations are not used in the system.

- Messages with a reliability of anything other than assured persistent may be lost or duplicated under certain conditions. For more information, see “Message reliability levels - JMS delivery mode and service integration quality of service” on page 635. If this is regarded as disrupting the order of messages, then only assured persistent messages should be used with a strictly ordered destination.
- Ordering of messages is assured within message priority, it should be understood that messages of a different priority can overtake messages of a lower priority.
- Ordering of messages is assured within message reliability, it should be understood that messages of one reliability can overtake messages of a different reliability.
- Although multiple consumers are not allowed to attach to an ordered destination, multiple producers can be used to send messages to ordered destinations. Ordering across multiple producers is not assured as messages appear at the destination in the order that the sending transactions are committed.
- Application code might contain logic that can disrupt message ordering.

Message selection and filtering

Message selection and filtering can occur when a consumer attaches to a destination.

When attaching to a destination, a consumer can provide a “selector”, a filter expression indicating the messages it is prepared to accept. The filter expression is a predicate referencing named fields in the message header or body. A particular instance of this occurs in the field of publish/subscribe, where subscribing applications typically express an interest in one or more topics and the selector is then a filter on the topic field carried in a message header. In JMS 1.1, selector expressions are restricted to referencing fields in the JMS header and the JMS properties of the JMS message; there is no access to the message body.

Using the default messaging provider, a topic is an identifier that a producer puts in the header of a message to enable message selection by consumers. The topics are grouped into topic namespaces, which have a tree-like hierarchical structure with a single root. This allows a subscriber (consumer) to connect with a wildcard selector that matches on an entire topic namespace, or a subtree of that namespace.

Message stores

Message stores are important in the operation of messaging engines. To host queue-type destinations, a messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them, or preserve messages in case the messaging engine fails. Each messaging engine has one and only one message store. This can either be a file store or a data store.

A message store enables a messaging engine to preserve operating information and to retain those objects that messaging engines need for recovery in the event of a failure.

A messaging engine preserves both volatile and durable data in its message store. Volatile data is lost when a messaging engine stops, in either a controlled or an uncontrolled manner. Durable data is available after the server restarts. For more information, see “Message reliability levels - JMS delivery mode and service integration quality of service” on page 635. A messaging engine stores various types of data, including messages, transaction states, and communication channel states.

When started, a messaging engine obtains configuration information from the WCCM (WebSphere Application Server Common Configuration Model) repository. A messaging engine retrieves all other data from its own file store or data store.

Important: There are currently no facilities available for migrating from a data store to a file store.

- “Relative advantages of a file store and a data store” on page 574

- “File stores”
- “Data stores” on page 579
- Message store high availability

Relative advantages of a file store and a data store

You must decide whether to use a file store or a data store for your messaging engine, by considering the advantages of each type.

Using a file store rather than a data store for your messaging engine can have several advantages:

Better performance
: To achieve best performance using a data store, you often have to use a separate remote database server. With file store you can achieve even better performance without having to use a separate remote database server.

Low administration requirements
: The file store combines high throughput with little or no administration. This makes it suitable for those who do not want to worry about where the messaging engine is storing its recoverable data. File store improves on the throughput, and scalability of Apache Derby.

Lower deployment costs
: Use of data store might require database administration to configure and manage your messaging engines. File store can be used in environments without a database server.

Some organizations prefer to use data store because it uses their existing resources more effectively. For example, this might be the case for a company with a strong team of database specialists, or a stable database infrastructure.

If there is a transient connectivity loss to the file system, the application server must be restarted once the connectivity to the file system is restored. Whereas, in the case of the data store, the messaging engine can recover from the database itself. In such situations, the data store will be a preferred high availability option than the file store system.

IBM i Data store has the following technical advantage over file store: with data store, some Java EE applications can share JDBC connections and benefit from one-phase commit optimization. For more information see Sharing connections to benefit from one-phase commit optimization. File store does not support this optimization.

Data stored in both data store and file store benefit from security features provided by WebSphere Application Server when accessed using the WebSphere Application Server APIs, that is when using JMS messaging. Further security features are available depending on the type of message store you use. For more details see “File stores” and “Data stores” on page 579.

- Data store: you access your chosen database by using a userid and password that is administered using the supplied tools for your specified DBMS. Logical and physical separation of your database server can also be used to improve the overall security of your data.
- File store: additional security can be provided when using a file store by careful consideration of your file store files. For example, using a secure network-attached drive to store your file store files improves the physical security of your data. Another example is storing your files on an operating system encrypted file system.

File stores

File stores use a file system to preserve operating information and to persist the objects that messaging engines need for recovery in the event of a failure.

A file store is a type of message store that directly uses files in a file system through the operating system. The data storage in a file store is split into three levels: the log file, permanent store file, and temporary store file.

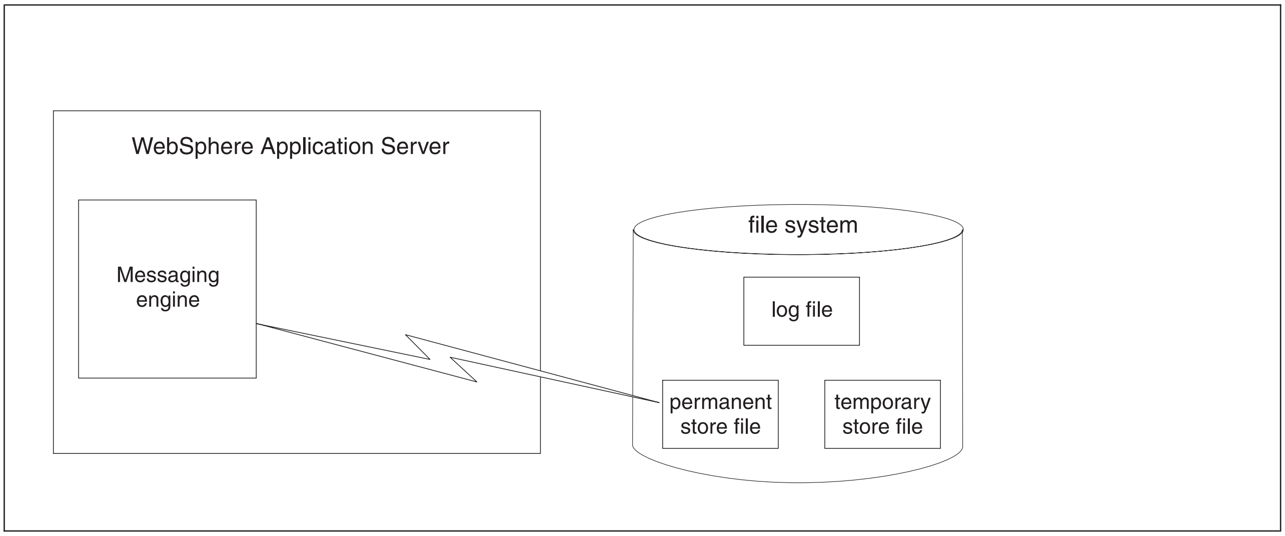


Figure 84. The relationship between a messaging engine and its file store

Log file

This file contains information about currently active transactions and data that is not yet written to a store file. It is a circular log and its file size is static while a messaging engine is running, but can be changed if required. A restart of the messaging engine is required for the changes to take effect. The size of the log file limits the maximum size of a message that can be sent.

Permanent store file

This file contains permanent data that is retained after the restart of the messaging engine, such as persistent messages, queue data, and information about the storage and transmission of persistent messages.

The permanent store file can be configured to have a maximum and minimum size, or to be unlimited in size. The file can grow from the minimum size (or as required in the unlimited case) but will never shrink (even if the maximum size is set lower than its current size). The file sizes can be changed in the administration console, but a restart of the messaging engine is required for the changes to take effect.

Similar to a file system, when data is deleted from the store, the data in the file is not deleted, only the directory information is updated. This means that if a message is consumed, the message data may still be present in the store file, but the directory information that includes this data in the store is updated to reflect the fact that it is deleted.

Temporary store file

This file contains temporary data that is not retained after the restart of the messaging engine, such as nonpersistent messages that were spilled to the file store to release memory from the JVM heap. The temporary store file contents are truncated when the messaging engine starts.

The permanent store file can be configured to have a maximum and minimum size, or to be unlimited in size. The file can grow from the minimum size (or as required in the unlimited case) but will never reduce (even if the maximum size is set less than its current size). The file sizes can be changed in the administration console, but a restart of the messaging engine is required for the changes to take effect.

Similar to a file system, when data is deleted from the store, the data in the file is not deleted, only the directory information is updated. This means that if a message is consumed, the message data may still be present in the store file, but the directory information that includes this data in the store is updated to reflect the fact that it is deleted.

You can configure where the file store files must be placed. By default, the file store uses a subdirectory in the following path: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/${ME_NAME}`. The file store directory contains two other directories; the log directory that contains the log file and the store directory that contains both the PermanentStore and TemporaryStore files.

Hints and tips for configuring file store size

Preserving the appropriate amount of space within the log file, permanent store file, and temporary store file of a file store helps to ensure that operations and transactions behave as expected.

The default settings for the minimum and maximum sizes are most suitable if you are not using a dedicated disk for your file store because they protect the file store from other disk users, and also protect other disk users from the file store. The default settings are less appropriate if you have a disk that is dedicated to file store use, and in this case you might want to consider setting the maximum size set to unlimited.

Input/Output performance typically poses a significant challenge for file stores. Performance improvements can be achieved using dedicated disks for the file store to avoid contention.

If your file store is not on a dedicated disk, consider making the minimum and maximum sizes the same. Initially, the message store reserves the amount of space defined by the minimum setting. Therefore, if you use the same value for the maximum and minimum settings, it reduces the likelihood of other applications using the disk space that you intended for the file store usage. Setting the maximum size to a value other than unlimited reduces the likelihood of the file store using disk space that you intended for other applications usage. Other applications can include, for example, the application server itself.

The most significant performance gains can be derived by using disk hardware configured to ignore the flags set by the operating system to indicate that a synchronous write is required when a reliable battery backed storage device is used. Such a device can effectively maintain the logs in memory and store files so that the writes do not have to wait for physical disk movements. Rather the data written is stored in a cache, and the assurance that the data is not lost in the event of a failure comes from the disk vendor on the basis that the cache is reliable either due to the battery or some other reason.

Log file size

Data is first written to the log file sequentially, that is, new records are added to the end of the file. When the end of the log file is reached, old records at the beginning of the log file are overwritten by new records and this process repeats. Later, the data is written to the permanent store file and temporary store file. The exception is a short-lived data, which is only written to the log file.

The log file size lays the limitation on the maximum size of messages that can be sent. A LogFileFullException is thrown if the log file is too small to handle large messages. The exception occurs because a single large message or multiple large messages are sent before the file store moves them to the store files. The size of the log file needs to be increased if this exception is encountered; otherwise, the default size is sufficient. Only testing using the maximum message size and load expected can accurately determine whether the default size is too small. It is not always beneficial to simply allocate a larger log file size because the size of the log file is used by algorithms that control the internal workings of the file store. An overly large log file results in increased memory usage.

Store File size

The permanent store file and temporary store file each have a minimum reserved size and a maximum size. When they are created, the permanent and temporary store files consume their minimum reserved sizes, plus the size of the log. If the maximum size is set to a larger value than the minimum reserved size, the files grow up to the maximum size as required.

The default store file sizes are sufficient. However, testing only with specific configurations can reveal what size store file might be required. This testing involves filling each queue point with messages of the largest expected size (until the message high limit threshold is reached) when the store file is set to unlimited. The store file size reached during this test, plus the log file size is usually a safe size to use as the maximum store file size.

The store file maximum size is not intended to be the limit that prevents further messages being sent. This is because it is not only message data that is stored in the store file. Queue data and protocol state data is also stored in the store file. Instead, the high threshold of the queues must be used to prevent any more messages being sent.

Once a store file is full, there is a remote chance that the store might not be able to receive more data even if all the messages in the store are consumed. This can happen if the log file size is over half the size of the maximum store file size. A rule of thumb to minimize the occurrence of this is to ensure that the minimum store file size is double the log file size and that the maximum is twice the minimum plus the size of the log file.
For example, if the log file size is 100 MB then the minimum store file size can be 200 MB (double the size of the log file), and the maximum size can be 500 MB (double the size of the minimum store file plus the log file).

The default configuration for file store attributes is intended to be sufficient to be used in typical messaging workloads without further administration. To improve the performance or availability of the log file or the two store files, you can configure the file store attributes to control where these files are placed. Similarly, you can configure the attributes that control the sizes of the log file and two store files, for example to handle workloads with many active transactions, large messages, or a large volume of message data resident in the messaging engine.

Note: This method of improving performance cannot be guaranteed on a compressing file system, for example, on an NT file system with the **Compress this directory** option selected. You must avoid configuring a file store to use a compressing file system for production use.

The following table shows the minimum and default values for file store attributes.

Table 44. File store attributes and values. The first column lists the file store attributes. The second column provides the description of the attributes. The third column provides the minimum and default values of the attributes.

Attribute	Description	Minimum and default values
Log size	Size of the log file, in megabytes	• *Minimum*: 10 MB • *Default*: 100 MB
Minimum permanent store size	The minimum number of megabytes reserved by the permanent store file. **Note:** The minimum store file size must always be at least as big as the log file.	• *Minimum*: 0 • *Default*: 200 MB
Maximum permanent store size	The maximum size in megabytes of the permanent store file. **Note:** Consider making the maximum store file size double the size of the log file. The maximum size of the store file must always be larger than the log file. If the log file size is the same as the maximum store file size then the messaging engine will not start.	• *Minimum*: 50 MB • *Default*: 500 MB
Minimum temporary store size	The minimum number of megabytes reserved by the temporary store file. **Note:** The minimum store file size must always be at least as big as the log file.	• *Minimum*: 0 • *Default*: 200 MB
Maximum temporary store size	The maximum size in megabytes of the temporary store file. **Note:** Consider making the maximum store file size double the size of the log file. The maximum size of the store file must always be larger than the log file. If the log file size is the same as the maximum store file size then the messaging engine will not start.	• *Minimum*: 50 MB • *Default*: 500 MB

Table 44. File store attributes and values (continued). The first column lists the file store attributes. The second column provides the description of the attributes. The third column provides the minimum and default values of the attributes.

Attribute	Description	Minimum and default values
Unlimited permanent store size	Indicates whether the permanent store file is unlimited in size	• *Default*: false
Unlimited temporary store size	Indicates whether the temporary store file is unlimited in size	• *Default*: false
Log directory	Name of the directory that contains the log file	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>-<me_uuid>/log`
Permanent store directory	Name of the directory that contains the permanent store file	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>-<me_uuid>/permanentStore`
Temporary store directory	Name of the directory that contains the temporary store file	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>-<me_uuid>/temporaryStore`

File store exclusive access

Each file store contains information that uniquely identifies the messaging engine that created it. A file store can only be used by the messaging engine that created it.

The messaging engine opens the files in the file store in exclusive mode to prevent multiple instances of the same messaging engine from simultaneously using the file store, for example, by accidental activation on the messaging engine of multiple servers in a cluster. When a messaging engine stops, either in a controlled fashion or as a result of server failure, the file store files are closed. Subsequently another instance of the same messaging engine is able to open the file store.

File store disk requirements

The reliability of your storage infrastructure affects the ability of WebSphere Application Server to maintain the integrity of your data.

Consult the documentation for your storage infrastructure for information on the level of reliability that it can be configured to provide. Examples of components which might be included in your storage infrastructure are: hard disk drives, RAID controllers, file systems, and network file system protocols.

Input/Output Reliability

The log and store files are written using Java APIs in such a way that requires the JVM to either set flags to indicate that all writes are synchronous or all previous writes must be forced to the disk after certain API calls. This requires that the occurrence of any failure will not result in data written in a synchronous write or before a force being lost, corrupted, or written out of order.

A number of layers are involved in honoring these write requests. Only the operating system, file system, and hard disk vendors can indicate whether the various configurations available will provide the level of reliability required. This is also true of other logging systems, such as databases.

In particular, if the storage device is not on the local machine where the messaging engine is running, and the files are residing on a network file system such as NFS, then NFS must be configured to ensure that these requirements are met. There is no test to ensure that write caching is not taking place.

File Location

If a messaging engine is in a cluster bus member, then it can run on different servers. This requires the file store files to be located on shared storage. The path to the file store files, as configured in the administration console, must be a path to the same files on each machine where the messaging engine can run. This can be achieved using NFS or some other advanced storage mechanism.

If the files in the specified path are not the same files, then when the messaging engine fails over from one server to another, it is effectively a different messaging engine with the same name. None of the persistent data is available to the new instance of the messaging engine.

File Locking

The log file is locked using java.nio.channels.FileLock.tryLock(). It is mandatory for the operating system and file system to honor this lock in all cases, and that the lock is released when requested explicitly or the Java process in which the messaging engine is running terminates unexpectedly.

Specifically, the lock if held, must prevent any other process from locking the file, even if that file is being accessed by a different machine. In addition, if the Java process running the messaging engine terminates unexpectedly, the lock must be released so that the other machine can access the file.

This requirement facilitates the case where a messaging engine is in a cluster bus member, and so can run on multiple different servers. If the HAManager directs a messaging engine to start because the first instance of the messaging engine failed, then it must be able to lock the log file if the other process has ended. Equally, if the other process has not ended (the so called split brain scenario), then the new instance of the messaging engine must not be able to acquire the lock, thus preventing the messaging engine from starting.

A file locking test tool can be used to ensure that a file system does provide the basic locking requirements. The IBM Support document contains more details about the file system locking protocol test tool.

Data stores

A data store is a message store that uses a relational database. A messaging engine uses a data store to store operating information in the database, as well as to preserve essential objects that the messaging engine needs for recovery in the event of a failure.

A data store consists of the set of tables that a messaging engine uses to store persistent data in a database. See Data store tables for a list of the tables that comprise a data store. All the tables in a data store are held in the same database schema. You can create multiple data stores in the same database, provided that you use a different schema name for each data store.

The one-to-one relationship between a messaging engine and a data store means that every messaging engine must have its own data store. A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine. Figure 85 on page 580 illustrates these relationships.

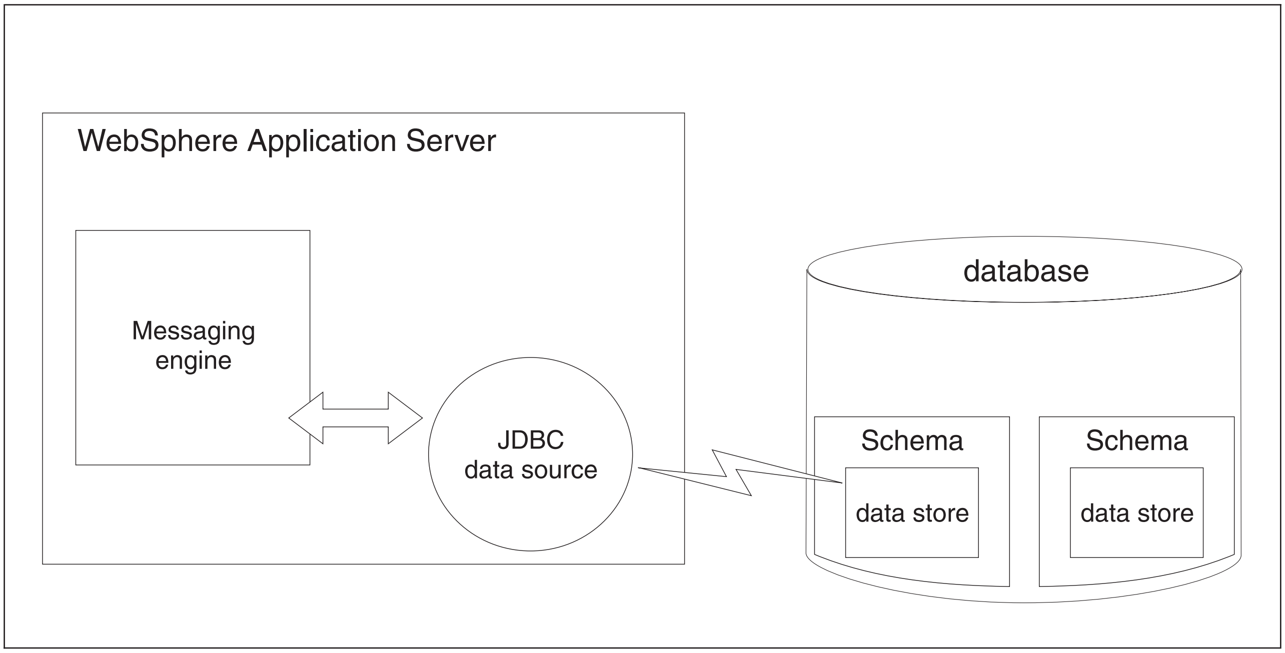


Figure 85. The relationship between a messaging engine and its data store.

All the tables in the data store must be stored in the same schema. You can create more than one data store in a database, provided that you use a different schema name for each data store. Although every messaging engine uses the same table names, its relationship with the schema gives each messaging engine exclusive use of its own tables.

Data store topologies

You have several options for the relative location of a data store and its messaging engine. The topology also defines the relationship of a data store with other data stores.

The following options affect your choice of data store topology:

- The data store can either run on the same node as its messaging engine or on a remote node.
- The data store can either have a dedicated database or it can share a database with other data stores.

Tip: If you are using the Informix RDBMS, configure a separate database instance for each messaging engine.

Data store life cycle

Starting or deleting a messaging engine affects the life cycle of its data store. Appropriate actions must be carried out on the data store.

Starting the messaging engine

When you start a messaging engine, it performs checks on the tables that comprise the data store to determine whether they are suitable. If the tables do not exist, and you have selected the **Create tables** option when configuring the messaging engine, the messaging engine attempts to create the tables. If you have not selected this option, your database administrator must create the tables beforehand, by using the data definition language (DDL) statements generated by the `sibDDLGenerator` command.

Make sure that the database that contains the data store is available before starting the messaging engine, or the server that is hosting the messaging engine. If the database is unavailable for more than 15 minutes, the messaging engine cannot connect to the data store, fails to start, and might enter the stopped state. You must restart the application server to start the messaging engine.

Stopping the database

If you want to stop the database that contains the data store, ensure that the messaging engine is stopped first. If the messaging engine is running and has exclusive locks on the data store, stopping the database can cause the messaging engine to be in an inconsistent state, because the messaging engine might continue to run and accept work. The same behavior occurs if the database fails while the messaging engine is running.

You can configure the messaging engine and its hosting server to shut down and restart when the database connection is lost, to prevent such inconsistencies. To configure this behavior, set the sib.msgstore.jdbcFailoverOnDBConnectionLoss custom property on the messaging engine. You can also tune your system to decrease the probability of the messaging engine failing to start before the database becomes available.

Removing a messaging engine

When you remove a messaging engine, WebSphere Application Server (base) does not delete the data store tables automatically. If you want to re-create the same messaging engine, you must first delete the previous set of tables. If you create a messaging engine with existing tables, these tables must be empty, so that the messaging engine can function correctly. Refer to the documentation for your chosen relational database management system (RDBMS) for information about how to delete tables. However, if you have created a data store with default settings, you do not have to delete previous tables.

Data store exclusive access

A messaging engine establishes a lock on its data store so that it has exclusive access to the data stored within. This process ensures the integrity of the data within the data store. Note that several factors can affect exclusive access locking in practice.

Each messaging engine establishes an exclusive lock on its data store. While the messaging engine is running, it maintains that lock to ensure the integrity of the data within the data store.

The data store lock uses database locks and so is an intrinsic part of the data store itself. The relational database management system (RDBMS) relies on the JDBC infrastructure or underlying TCP/IP protocol to indicate the failure of a messaging engine or application server. The RDBMS can then automatically release the data store locks in the database.

The SIBOWNER table in the data store holds the lock as a pair of unique identifiers in a single row. When it starts, a messaging engine uses these two identifiers to acquire and maintain its exclusive lock:

MEUUID
The unique identifier for a messaging engine, which remains the same whenever the messaging engine stops and restarts.

INCUUID
The incarnation identifier for a messaging engine, which changes every time the messaging engine starts.

These identifiers determine which messaging engine is using a data store. These identifiers also determine whether a running instance of a messaging engine has maintained its exclusive lock for the time period during which it was running.

The other table used in the data store locking process is the SIBOWNERO table. This table is used for locking only and stores no data in its one EMPTY_COLUMN column.

As a messaging engine starts, it first obtains an exclusive table lock on the SIBOWNERO table. The messaging engine then obtains an exclusive table lock on the SIBOWNER table and proceeds to check the contents. The messaging engine expects to find a single row of data in the table, or no data. If there is an existing row, the messaging engine starts only if the MEUUID that it finds matches its own unique identifier. If the MEUUID does not match, the messaging engine writes error message CWSIS1535 to the server error log and the messaging engine fails to start.

If there are no rows in the SIBOWNER table, the messaging engine inserts one row containing its own MEUUID and INCUUID. If there is a single row in the table and the MEUUID matches the unique identifier for the messaging engine, the messaging engine updates the INCUUID with its own incarnation identifier and attempts to obtain a new shared table lock on the SIBOWNER table. The messaging engine checks again that the INCUUID still matches its own incarnation identifier, in case another instance of the same messaging engine on another server is starting at the same time. If the INCUUID in the table does not match, the messaging engine fails to start.

The shared table lock on the SIBOWNER table remains until the messaging engine stops, with periodic refreshes to ensure that the lock is still in force. The exclusive table lock on the SIBOWNERO table is then released. If the messaging engine cannot refresh the lock on the SIBOWNER table, or discovers from the data in the SIBOWNER table that another messaging engine incarnation is using the data store, the messaging engine writes messages CWSIS1594 and CWSIS1519 to the server error log and the messaging engine stops immediately.

Data store performance

The performance of a data store, chosen as the message store for a messaging engine, can be affected by several factors such as providing the Java data base connectivity (JDBC) data source with sufficient connections. Use this information to help configure an optimal environment for a data store.

The workload that the messaging engine imposes on the relational database management system (RDBMS) is slightly different from usual database workloads, because the messaging engine performs mainly `SQL INSERT` and `DELETE` operations. Take this workload into consideration when using the tuning guidelines supplied by your RDBMS provider. The following information might also assist your database administrator.

Each messaging engine can request a large number of concurrent connections to the database. By design, a messaging engine uses many threads to perform database updates concurrently. Consider configuring the connection pool for the JDBC data source used by the data store with sufficient connections to cope with peak workloads. Your database administrator might have to change the database configuration to support this number of concurrent connections from the application server.

The performance of the data store influences messaging throughput to a significant extent. Throughput is usually limited by the write latency of the database log. Consider placing the database log on a fast, dedicated device that is configured for optimal write performance. Ideally, use a storage controller with a battery-backed memory cache so that the effects of disk rotation speed and seek latencies are not evident. For all except entry-level systems, use a RAID controller.

DB2 tip: To get the best performance from messages in the 3 KB to 20 KB range, you should consider putting the SIBnnn tables into a tablespace with 32 KB pages and adjusting the column width of the VARCHAR column to 32032 bytes.

Configuration planning for a messaging engine to use a data store

You must consider a number of choices before you configure a messaging engine to use a data store.

Relational database management system (RDBMS) for the data store

You might want to choose the RDBMS that you use for other applications, particularly if you are already familiar with the tools you use for managing that RDBMS. You might also want to consider the following criteria:

- Performance
- Scalability

- Availability, especially if you are running messaging engines in a high availability environment

When a new messaging engine that uses a data store is created on a single server, it is configured to use an Apache Derby data source by default. This enables the messaging engine to run without needing any additional configuration. The default embedded Derby data source is sufficient for many purposes. Other relational database management systems offer more comprehensive tooling and improved performance, particularly scalability on larger machines with more than two processors.

Note: WebSphere Application Server supports direct customer use of the Apache Derby database in *test* environments only. The product does not support direct customer use of Apache Derby database in *production* environments.

Database topology

You must consider several options when selecting the relative location of a data store and its messaging engine:

- Decide whether the data store will run on the same node as its messaging engine, or on a remote node. In some cases, running the data store on a remote node can improve performance. In other cases, a local database provides performance equivalent to a remote database. You might want to conduct your own performance analysis, because the performance characteristics can be very sensitive to the hardware specification.
- Decide whether the data store will have a dedicated database, or share a database with other data stores.

Automatic creation of database tables

Consider whether you want WebSphere Application Server to create the data store tables automatically or whether you want your database administrator to create the tables beforehand:

- WebSphere Application Server can create the data store tables automatically if you select the **Create tables** option when you configure the data store to use a data source. If you want to choose this option, you must first ensure that WebSphere Application Server has sufficient authority to create tables and indexes by setting up the required privileges for your chosen database.

  DB2 for z/OS restriction: The option for WebSphere Application Server to create the tables is not available with DB2 for z/OS. If you use DB2 for z/OS, your database administrator must create the data store tables manually.
- To enable the database administrator to create the tables manually, you must provide data definition language (DDL) statements created by using the `sibDDLGenerator` command.

Amount of BLOB space required to hold message data

Message data is stored in a database table column of datatype BLOB. Before you create a data store, you must consider the size of your expected workload to ensure that your database administrator creates a sufficiently large BLOB space to hold your message data.

Service integration security

Messaging security ensures that service integration bus users are authenticated, resources are protected by security checks, and messages are secured when they are in transit. Use these topics to learn how to secure the service integration bus and protect messages that are sent and received.

Security covers all of the following areas:

- Authenticating and authorizing users that attempt to connect to a bus, and use its resources.

- Securing communication transports between clients and messaging engines, and between messaging engines themselves.
- Authenticating peer messaging engines in a bus.
- Protecting the message store with a user identity.

When a bus is created with bus security enabled, the following conditions apply:
- The bus requires client authentication.
- The bus enforces authorization policy.
- The bus requires use of SSL transport chains.

You can use secure transport connections to ensure the confidentiality and integrity of messages that are in transit between application clients, the bus, and between messaging engines. This is achieved by defining transport chains and then referencing the transport chain name as follows:
- For application client connections: from the connection factory administered objects.
- For connections to foreign buses: from the **Target inbound transport chain** property of the service integration bus link.
- For connections to WebSphere MQ: from the **Transport chain** property of the WebSphere MQ link.
- For connections between messaging engines: from the **Inter-engine transport chain** property of the bus.

For more information, see “Secure transport configuration requirements” on page 546.

Note: When a secure bus is created, only SSL protected messaging chains are permitted. For example, you can use the InboundSecureMessaging transport chain.

In the routing properties for the service integration bus link for a foreign bus connection, the user ID applied to messages entering or leaving the foreign bus can be replaced by values specified by the Inbound user ID and Outbound user ID properties.

The ability to authenticate access to a foreign bus is provided by the **Authentication alias** property of the service integration bus link. You can specify an authentication alias at each end of the service integration bus link between two secure buses when you create each foreign bus connection. The user ID you specify in the authentication alias on each side of the link must be the same for authorization purposes. For example, consider a scenario where two messaging engines are connected by a service integration bus link. Messaging engine A presents the user ID and password to messaging engine B so that messaging engine B can authenticate messaging engine A. For details about creating a foreign bus connection, and therefore a service integration bus link, see Configuring foreign bus connections.
- “Service integration security planning” on page 585
- “Messaging security and multiple security domains” on page 587
- “Messaging security” on page 587
- “Security event logging” on page 588
- “Messaging security audit events” on page 588
- “Client authentication on a service integration bus” on page 591
- “Role-based authorization” on page 592
- “Destination security” on page 593
- “Mediations security” on page 594
- “Topic security” on page 595
- “Access control for multiple buses” on page 597
- “Message security in a service integration bus” on page 598

Service integration security planning

When you are planning the security of your messaging system, the range of options available to you can be described through a set of frequently asked questions.

These are some of the questions you might have when you start planning for messaging security:

- How do I secure the bus?
- Who has authority to access the bus?
- Who has authority to access bus destinations?
- Which connections must I secure?
- Which user IDs are stored in messages that flow between the bus and any foreign buses?
- What level of data store security do I need?

If you are using service integration with web services, refer to Securing bus-enabled web services.

How do I secure the bus?

Providing WebSphere Application Server global security is enabled, and a user registry is configured, a newly created service integration bus is secured by default. The **Enable bus security** flag in the bus definition is checked by default. This means that the messaging engine authenticates all connecting client applications, and performs authorization checks on every client application that attempts to access bus resources. For more information about enabling bus security, see “Messaging security” on page 587.

When you create a new bus, or you want to secure an existing bus, the Bus Security wizard prompts you to specify a security domain. The security domain contains the security settings for the bus. You can specify the default global domain, or an alternative domain, depending on the versions of the bus members:

Global domain

This is the default security domain. You must specify the global domain for mixed-version buses.

Cell level and custom domains

If the bus contains WebSphere Application Server Version 7.0 or later bus members only, you can configure the bus to use either the cell default security domain, or a custom security domain. A custom security domain typically contain security settings specific to a particular bus. You can configure additional, separate security domains for user applications such as the UserRepository. For more information about using multiple security domains for the bus, see “Messaging security and multiple security domains” on page 587.

For more information about how to add a new secure bus, see Adding a secured bus. If you want to secure an existing bus, see , and Securing an existing bus by using multiple security domains. If you want to migrate an existing secure bus from a global security domain to a cell level or custom security domain, see Migrating an existing secure bus to multiple domain security.

Who has authority to access the bus?

When a client application attempts to connect to the bus, the messaging engine authenticates the credentials (an identity and password) for the client application against the user registry. If the client authenticates successfully, the messaging engine checks that the client has authority to connect to the bus. Every client applications that has a valid user identity and password in the user registry can authenticate successfully, but you might not want every client application to have authority to connect to the bus. To control access to the bus, you must grant authority to specific client applications in the bus connector role for the bus. You create a group in the user registry, add the identities of the client applications to the group, and then add the group to the bus connector role for the bus. For more information, refer to Administering the bus connector role.

Who has authority to access bus destinations?

For each destination, you must decide which clients require authority to undertake operations at a bus destination, and which operations (or roles) they have to undertake. Authority is granted by adding groups and group members to roles. For example, if you want a group of client applications called *MyGroup* to send messages to a queue destination called *MyQueue*, you add *MyGroup* to the sender role for *MyQueue*. For more information, refer to Administering destination roles.

You can define a set of default permissions that apply to every destination in a bus. For example, if you want to authorize all the members of a group called *MyMediations* to send messages to every destination on a selected bus, you can add *MyMediations* to the default sender role. By default, all local destinations inherit roles from the default resource. You can choose to override default inheritance for selected destinations. For more information about default roles, see Administering default roles. For more information about overriding default role inheritance, see Disabling inheritance from the default resource.

If a group of client applications publish and subscribe to topics, the topics exist in a topic space. The identities of all the clients that publish to a topic to must belong to a group that has the sender role for the topic space. All the client applications that subscribe to a topic must belong to a group that has the receiver role on the topic space. For more information, see Administering topic space root roles. By default, there are also checks on authorization permissions at the topic level. You can disable the topic level check, or decide which groups of client applications you want to authorize to access selected topics.

Which connections must I secure?

Decide which of the following connections to secure with SSL:

- Connections between the clients and the servers (messaging engines).
- Connections between messaging engines within a bus.
- Connections between buses.

For more information, refer to Securing messages between messaging buses.

Which user IDs are stored in messages that flow between the bus and any foreign buses?

When a message is sent, the user ID of the sender is stored in the message and is used for any subsequent access control checks on the message. Where a link exists between buses, you can configure the Inbound ID and the Outbound ID on the link to control which user ID is stored in messages that flow between local and foreign buses.

The Inbound ID replaces the user ID in every message that flows across the link into the bus. The Inbound ID is used to control access to messages within the bus. You might want to configure an Inbound ID for the following reasons:

- The local bus and the foreign bus exist in separate security domains.
- The foreign bus is not secure.
- You can manage access control more easily when every message has the same user ID.

The Outbound ID replaces the user ID in every message that flows across the link out of the bus. You might want to configure an Outbound ID when you to prevent the original user ID from being carried in the messages on the foreign bus.

For more information, refer to Securing links between messaging engines.

What level of data store security do I need?

The messaging system can use a data store (database) to store messages on a disk. The messages in the data store might be protected by a username and password. You should consider whether this is sufficient security for your data store. Your data base might provide additional security options, for example data encryption. For more information, refer to Securing database access.

Messaging security and multiple security domains

When you secure a service integration bus, you assign it to a security domain that contains a set of security attributes. There are three types of security domain: global, cell level and custom. The type of security domain you use for a particular bus depends on your security requirements, the bus topology, and the versions of the bus members.

Global domain

This is the default security domain, and contains the administrative security settings.

You must assign the bus to use the global domain if the following conditions apply:

- The bus contains a WebSphere Application Server Version 6 bus member, or might contain a Version 6 bus member in the future.
- The bus is used for administrative purposes, and must share the administrative security settings.

You might also choose to use the global security domain if you have a simple bus topology, and have no need to use multiple security domains.

Cell level domain

Assigning the bus to the cell level domain enables the bus to use multiple security domains.

You might want to assign the bus to use the cell level domain if one of the following scenarios apply:

- Your company security policy requires that the administrative user repository is separate from the customer user repository. Using the cell level domain enables you to configure multiple sets of security attributes for administrative and user applications within a cell environment.
- For ease of configuration and maintenance, you want the bus, its user applications, and servers to share a common security configuration that is separate from the administrative security settings.

Custom domain

You must assign the bus to a custom domain if the following scenarios apply:

- You want to guarantee that the bus and its user application can access the same user realm. In this case, the bus and the user applications use the same custom domain.
- You want the bus to use a user realm that is dedicated to messaging, and have a separate user repository each for administrative and customer accounts.
- You want the bus, and each of its user applications in separate domains. The application users can interact with the users of the bus domain, which acts as a bridge between the application domains. In this case, only the bus requires information about the users in each domain .

Messaging security

Messaging security protects the service integration bus from access by unauthorized users.

Client authentication

When a client application attempts to connect to a messaging engine, the client provides credentials to the server, and they are authenticated against the user registry. If the credentials are found in the user registry, the client application authenticates successfully, and proceeds to the authorization checks. If a Secure Socket Layers (SSL) connection is configured, JMS client applications can authenticate by using client SSL. This removes the need for the client to specify a user ID and a password.

Authorization

When a connecting client application authenticates successfully, the messaging engine checks for authority to connect to the bus. Bus authorization is role-based. Specific roles are defined for each bus resource, and groups of users are added to roles. For example, if a client application belongs to a group that has been added to the bus connector role, the messaging engine grants the client application permission to connect to the bus. The messaging engine checks the set of roles defined for each bus destination to determine what action the client application can perform on the bus destination. By default, all local bus destinations can inherit a default set of roles. Inheritance of default roles can be overridden for a particular destination.

For publish subscribe, the messaging engine checks that the client has permission to access the topic space. If the Topic access check required attribute in the properties for a bus destination is set, the messaging engine additionally checks that client applications have permission to access the topic.

Transport encryption

Finally, the security administrator must ensure the confidentiality and integrity of messages in transit, by, for example, configuring an SSL secure transport for every bus connection.

Security event logging

Security events on a service integration bus are logged as audit or error records in the SystemOut.log file for the bus.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

A security event is the outcome of an attempt by a connecting client application to authenticate to a bus. There are two possible outcomes: authentication success, and authentication failure. Success is logged as an audit record, and failure as an error record. You can control the type and number of audit and error records logged for a bus by configuring a custom property for the bus, in the administrative console. The property is a name-value pair called `audit.bus.authentication` that can have one of the following three string values:

all Audit every attempt to authenticate to the bus.

failure Audit only failure to authenticate to the bus.

none Do not audit any attempt to authenticate to the bus.

Messaging security audit events

Messaging security audit events are produced when a messaging client sends or receives a message, or updates a service data object repository, and when a publisher submits a message to a topic.

To audit messaging security, you create security event type filters for the audit events produced during messaging operations. Details of the audit events produced are included in the following descriptions of auditable messaging operations:

- “A messaging client sending a message to a message destination” on page 589
- “A messaging client receiving a message from a message destination” on page 589
- “Messaging engines connecting to one another on the same bus” on page 589

- “Messaging engines connecting to one another on different buses” on page 590
- “A publisher sending a message to a topic” on page 590
- “A messaging client receiving a message from a subscription” on page 590
- “A cell administrator updates a service data object (SDO) repository” on page 591

A messaging client sending a message to a message destination

The messaging security audit events SECURITY_AUTHN, SECURITY_AUTHZ and SECURITY_AUTHN_TERMINATE are produced when a messaging client sends a message to a message destination.

Audit events produced when a messaging client sends a message to a message destination are as follows:

1. The messaging client connects to a messaging bus:
   a. A SECURITY_AUTHN event is produced when the identity of the messaging client connecting to the messaging bus is authenticated.
   b. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the bus.
2. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the destination queue.
3. One or more messages are sent from the client to the destination queue. No audit events are produced.
4. When the connection between a messaging client and a messaging bus is terminated, a SECURITY_AUTHN_TERMINATE event is produced.

To record these audit events, you create security event type filters for them.

A messaging client receiving a message from a message destination

Messaging security audit events SECURITY_AUTHN, SECURITY_AUTHZ and SECURITY_AUTHN_TERMINATE are produced when a messaging client receives a message from a message destination.

Audit events produced when a messaging client receives a message from a message destination are as follows:

1. The messaging client connects to a messaging bus:
   a. A SECURITY_AUTHN event is produced when the identity of the messaging client connecting to the messaging bus is authenticated.
   b. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the bus.
2. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the destination queue.
3. One or more messages are sent from the message queue to the client. No audit events are produced.
4. When the connection between a messaging client and a messaging bus is terminated, a SECURITY_AUTHN_TERMINATE event is produced.

To record these audit events, you create security event type filters for them.

Messaging engines connecting to one another on the same bus

The messaging security audit events SECURITY_AUTHN and SECURITY_AUTHZ are produced when a messaging engine connects to another messaging engine on the same bus.

Audit events produced when a messaging engine connects to another messaging engine on the same bus are as follows:

1. A SECURITY_AUTHN event is produced when the identity of the messaging engine connecting to the bus is authenticated by the destination messaging engine.
2. A SECURITY_AUTHZ event is produced when the identity of the messaging engine is checked for access authority to the destination messaging engine.

To record these audit events, you create security event type filters for them.

Messaging engines connecting to one another on different buses

The messaging security audit events SECURITY_AUTHN and SECURITY_AUTHZ are produced when a messaging engine connects to another messaging engine on a foreign bus.

Audit events are produced when a messaging engine connects to another messaging engine on a foreign bus as follows:

1. A SECURITY_AUTHN event is produced when the identity of the messaging engine connecting to the bus is authenticated by the destination messaging engine.
2. A SECURITY_AUTHZ event is produced when the identity of the messaging engine is checked for access authority to the destination messaging engine.

To record these audit events, you create security event type filters for them.

A publisher sending a message to a topic

The messaging security audit events SECURITY_AUTHN, SECURITY_AUTHZ and SECURITY_AUTHN_TERMINATE are produced when a messaging client sends a message to a subscription topic.

The audit events produced when a messaging client sends a message to a subscription topic are as follows:

1. The messaging client connects to a messaging bus:
   a. A SECURITY_AUTHN event is produced when the identity of the messaging client connecting to the bus is authenticated.
   b. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the bus.
2. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the subscription.
3. One or more messages are sent from the client and submitted to the subscription. No audit events are produced.
4. When the connection between a messaging client and a messaging bus is terminated, a SECURITY_AUTHN_TERMINATE event is produced.

To record these audit events, you create security event type filters for them.

A messaging client receiving a message from a subscription

The messaging security audit events SECURITY_AUTHN, A SECURITY_AUTHZ, and SECURITY_AUTHN_TERMINATE are produced when a messaging client receives a message from a subscription.

Audit events are produced when a messaging client receives a message from a subscription as follows:

1. The messaging client connects to a messaging bus:
   a. A SECURITY_AUTHN event is produced when the identity of the messaging client connecting to the bus is authenticated.
   b. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the bus.

2. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for authority to receive the message. This check is only performed once per connection session, and the result is cached.

   If changes are made to the access rights for the topic, or a new subscription is created that matches an existing topic (either exactly or as a result of a 'wildcard' match), the cache entry for the topic becomes invalid. Another check is made for authority to receive the message, and another SECURITY_AUTHZ event is produced.
3. One or more messages are received by the client from a subscription. No audit events are produced.
4. When the connection between a messaging client and a messaging bus is terminated, a SECURITY_AUTHN_TERMINATE event is produced.

To record these audit events, you create security event type filters for them.

A cell administrator updates a service data object (SDO) repository

The messaging security audit events SECURITY_AUTHN, SECURITY_AUTHZ, SECURITY_MGMT_CONFIG, and SECURITY_AUTHN_TERMINATE are produced when a cell administrator changes the contents of a service data object (SDO) repository in an import or remove operation.

Audit events produced when a cell administrator changes the contents of a service data object (SDO) repository in an import or remove operation are as follows:

1. The cell administrator connects to a messaging bus:
   a. A SECURITY_AUTHN event is produced when the identity of the messaging client connecting to the Enterprise Java Bean container that includes the SDO repository is authenticated by the application server.
   b. A SECURITY_AUTHZ event is produced when the identity of the messaging client is checked for access authority to the foreign bus.
2. A SECURITY_MGMT_CONFIG event is produced when messaging client issues a remove or import request that results in a change to the contents of the SDO repository.
3. When the connection between a messaging client and the foreign bus is terminated, a SECURITY_AUTHN_TERMINATE event is produced.

To record these audit events, you create security event type filters for them.

Client authentication on a service integration bus

When a client application attempts to connect to a messaging engine on a secure service integration bus, the client application provides credentials to the server that are checked against the user registry.

Client authentication is one security mechanism for protecting the bus from unauthorized access, alongside authorization, and transport encryption. Client authentication is effective only when administrative security is enabled on WebSphere Application Server, and messaging security is enabled on the bus.

A connecting client application provides credentials that the server verifies against the user registry. The following types of credential are permitted:

- User ID and password
- X509 certificate

The security administrator specifies the type of user registry when configuring administrative security.

WebSphere Application Server Version 6 supports different types of user registry, including federated repositories.

WebSphere Application Server Version 7.0 or later can use the user registry from the administrative domain, or the bus or cell domains.

The bus security administrator checks that the credentials for the connecting client are valid in the user registry for the cell hosting the bus. If the server is enabled to allow a JMS client application to use Secure Sockets Layer (SSL) client authentication, a stand-alone Lightweight Directory Access Protocol (LDAP) user registry is required.

When application code in an EJB or web container invokes the JMS client, and accesses it as a J2EE Connector Architecture (JCA) resource, authentication is determined by whether the application code has been configured to allow container-managed or application-managed sign-on to resources. For further details, see Java EE connector security.

If an application fails to authenticate, a `JMSSecurityException` is thrown.

Role-based authorization

Service integration messaging security uses role-based authorization. By adding and removing users and groups in access roles you can control who has access to a secured bus and its resources.

When bus security is enabled, you must add users and groups to access roles to grant them authority to connect to the bus, and to work with its messaging resources, for example a destination or a topic space. You can administer users and groups in access roles either by using the administrative console, or by using **`wsadmin`** reference commands.

Access roles

When you add a user to an access role, you grant that user all the security permissions contained within the role type. You can add users to the following access roles:

Connector role
Grants the user the permission to connect to the local bus.

Sender role
Grants he user the permission to send a message to a destination.

Receiver role
Grants the user the permission to receive a message from a destination.

Browser role
Grants the user the permission to browse messages on a destination.

Creator role
Grants the user the permission to create a temporary destination prefix.

Users and groups

Any user or group that you want to add to an access role must have a definition in the user registry. A user that belongs to a group that has been added to an access role is authorized to carry out the operations permitted for that role.

There are three special types of groups:

All Authenticated
Contains all authenticated users. If the All Authenticated group is authorized to undertake an operation, then all authenticated users are authorized to undertake it. When a bus is created, an initial set of authorization permissions is created that allows all users in the All Authenticated group access all local destinations. You can change these permissions to restrict access to the specific users and groups that you want to connect to the bus.

Everyone
Contains all users whether or not they are authenticated.

Server
Contains every WebSphere Application Server within a cell.

Messaging operations

When messaging security is enabled, all operations on the following resources require authorization:

Buses When a user connects to a local bus, the system checks that the user has authorization to connect to the bus. For a user who has already connected successfully to a local bus to send a message to a destination on a foreign bus, the user requires authorization to access the foreign bus.

Destinations
Users require authorization to undertake messaging operations (typically send, receive, and browse) on a destination.

Temporary destinations
A user must have the creator role to create a temporary destination. By default, the All Authenticated group have the creator role. When an authorized user (a client application) creates a temporary destination, a temporary destination prefix is specified. The messaging engine uses the temporary destination prefix at runtime to determine which operations the client application can perform. A client application that has the sender role for a temporary destination prefix is authorized to send messages to the temporary destination.

Topic spaces and topics
To access a topic within a topic space, a user must be authorized to access both the topic space, and the specific topics within this topic space. To make topic authorizations easier to manage, a topic inherits authorization permissions from its parent in the topic namespace by default. You can change inherited permissions for any given topic, or you can disable inheritance at the topic space level for a given topic space. In this case, the system checks that the user is authorized to access the topic space, but no further checks are made at the topic level.

Default authorization permissions

The default authorization permissions enable you to quickly grant access to all local destinations. Although the All Authenticated group has full access to all destinations, only the Server group has the bus connector role. If you want a particular user to access the bus, you must add that user to the bus connector role for the bus. When users have the bus connector role, they have full access to the bus.

The default permissions apply to all destinations in a local bus namespace, with the following exceptions:

- A destination for which inheritance is disabled
- Foreign destinations
- Alias destinations that have an alias bus name that is not the local bus name

Destination security

When messaging security is enabled, the authorization policy for resources on the service integration bus is enforced, and client applications must have authority to access bus destinations.

Destination authorization

Access to a destination is role-based. By assigning a group of users to a specific role for a specific bus destination, you authorize the group to undertake a specific operation on that bus destination. The roles for a destination depend on the type of destination:

Sender
This role applies to alias, foreignDestination, port, queue, and topicSpace destinations.

Receiver
This role type applies to alias, port, queue, and topicSpace destinations.

Browser
This role type applies to alias, port, and queue destinations.

Creator
This role applies to temporary destinations only.

You define destination role assignments on the bus that owns the destination. If a message is routed between two or more destinations, the members of a group require authority to access each of the destinations.

Temporary destination authorization

A temporary destination prefix, which is specified when a temporary destination is created, is used at runtime by the messaging engine to determine access authority for the temporary destination. When a temporary destination is created, the identity of its creator is assigned to the creator role for the temporary destination prefix. By default, all authenticated users can create temporary destinations. To grant the members of a group access to a temporary destination, you must assign the group to the sender role for the corresponding temporary destination prefix.

Multiple destinations authorization

When a message arrives at a destination, it might be routed onwards to one or more destinations before it is received by an application. This might happen, for example, if there is a mediation on the first destination. When the message arrives at the first destination, the messaging engine checks that the identity of the sending application has authority in the sender role to send messages to the destination, and adds the identity of the sender to the message. If the message is routed on to another destination, the messaging engine checks that the sender identity in the message has authority in the sender role for the destination to which it is being routed. The messaging engine continues to check the authority of the sending application along the forward routing path of the message until it arrives at a mediated destination. The mediation might (but does not always) replace the sender identity in the message with the mediation identity. If this happens, the messaging engine uses the mediation identity to check the authority of the sender to send to the next destination in the forward routing path, not the identity of the original sender.

Inheritance of security authorization

A destination can inherit role assignments from the default resource. Only role types that are permitted for a particular destination type can be inherited. For example, a queue destination can inherit the browser role from the default resource. Inherited role assignments are added to any existing role assignments for a destination. For example, a queue destination that has members of a group called Group 1 in the sender role, can inherit Group 2 in the sender role for the default resource.

Mediations security

When bus security is enabled, authorization permissions are required to ensure that mediations can run, and undertake messaging operations securely on a service integration bus. There are mechanisms for mediations security, and implications for running mediations on a bus that spans multiple security domains.

When bus security is enabled, the messaging engine must be authorized to access the mediation. Authorization is granted by using a mediations authentication alias or an LTPA token, depending on the version of the bus member:

- A WebSphere Application Server Version 7.0 or later bus member uses an LTPA token for messaging engine authentication. If an authentication alias is specified, it is used but a password is not required.

- A WebSphere Application Server Version 6 bus member requires an authentication alias to ensure that the mediation can be called. For more information, see Configuring the bus to access secured mediations.

When an application sends a message to the bus, the identity of the sender application is associated with the message. The message is sent to the next destination in the forward routing path providing the message originator has `Sender` authority for that destination. If a mediation processes the message in some way at the target destination, the identity associated with the message is preserved by default. You can program the mediation to reset the message identity to the identity under which the mediation code runs. For example, if the mediated destination represents the boundary between two security domains, the sender application is not authorized to access the mediated destination. By translating different identities into a single user identity, you can control access between security domains. For more information about programming mediations, see “Mediation programming” on page 605. For more information about using the resetIdentity() method, see SIMediationSession .

When you install a mediation for use when bus security is enabled, you must ensure that the identity that is used by the bus to call the mediation can access the mediation. By default, a mediation is unauthenticated. You can configure it to use the mediations authentication alias by specifying a RunAs role by using the assembly tools. For more information, see Configuring an alternative mediation identity for a mediation handler.

If bus security is enabled, and a mediation is sending messages to a destination, the mediation identity requires authority to access the destination. For more information, see Administering authorization permissions. Any new messages sent by the mediation are sent using the mediation identity.

If administrative security is disabled, an identity is not configured for the mediation. If bus security is enabled, and administrative security is disabled, the mediation is not authenticated to access bus destinations.

Using mediations in multiple security domains

You can run mediations successfully in a bus topology where the members of a bus span multiple security domains. The bus security configuration provides an option, called **`addUserServerIdForMediations`**, to allow mediations to run under a server identity. In this case, a mediation authentication alias is not required.

Mediations are deployed as applications, and run in the domain used by the application server, not the bus domain. Because the mediation authentication alias applies to the whole bus, if you run a mediation on multiple servers in different domains, you must ensure that the user identity in the mediation authentication alias exists in the configuration for each domain. Alternatively, you can choose to use the server identity option. You can use this option even if multiple domains are not in use.

Topic security

Client applications publish and subscribe to a topic. When messaging security is enabled, client applications require authority to access a topic.

Topics are contained within a topic space, which is a type of destination. Within the topic space, topics are organized into one or more topic hierarchies, based on the topic names. The topic hierarchies are joined to a *virtual root* that is created when the topic space is created. A topic is created within the topic space when a client application publishes to the topic.

When a connection accesses a topic, an access check is performed to ensure that the user associated with the connection has permission to access the topic space destination that contains the topic. A second check is performed to ensure that the user also has access to the topic itself, within the topic hierarchy owned by the topic space destination. This allows for finer-grained control of access to topics, as shown in the diagram later in this section. The diagram shows a topic space called *tspace1* that contains two topic hierarchies, called *sports*, and *cars*.

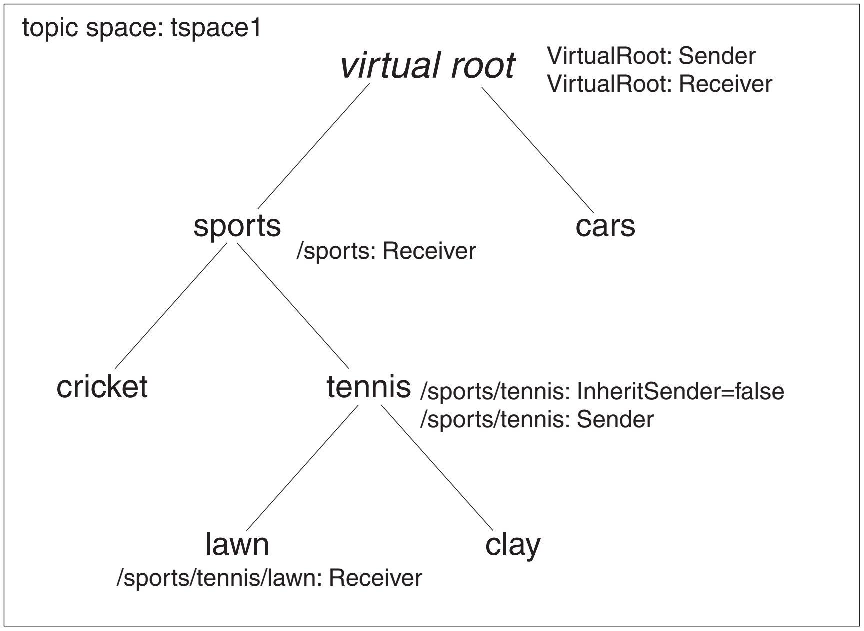


Figure 86. Example of a topic space with topic-level authorization

Note: A connection must have authority to access the topic space destination, regardless of any access it has been granted within the topic hierarchy owned by that topic space destination.

Each bus can contain more than one topic space. Each topic space is independent of other topic spaces on the bus. Topics in separate topic spaces are not related, even if they have the same name. Consider, for example, a bus that contains two topic spaces called *tspace1* and *tspace2*, and each topic space contains a topic called *cars*. If a client application subscribes to *cars* in *tspace1*, it can only receive messages published to the *cars* topic in *tspace1*.

Role-based authorization and topic inheritance

Topic security is based on role-based authorization. For further information, see “Role-based authorization” on page 592. To make it easier to administer security authorization for a large number of topics, the access role defined for a topic contains the access authority for the topic itself and, by default, for any topic beneath it within the topic hierarchy. A topic therefore inherits roles from the parent topic. An access role defined on the virtual root therefore contains, by default, access authority for every topic in the topic space. In the example given above, the sender and receiver roles have been defined on the virtual root of topic space *tspace1*.

You can define new roles for a topic, and you can disable topic role inheritance for any topic in the topic hierarchy. New roles are added to any roles that the topic inherits from its parent topic. The figure above contains the following examples of how topic role inheritance works:

- The topics *sports*, *cricket* and *cars* topics inherit the sender role defined on the virtual root, but inheritance of the sender role for the topic *tennis* is disabled. This means that the topic *tennis* and its children cannot inherit the sender role from the virtual root. The sender role is defined for the topic *tennis* itself. The messaging engine checks that a connecting client application has authority in the sender role for the topic *tennis* and its children.
- Every topic in the hierarchy inherits the receiver role from the virtual root. The *sports* topic also has its own receiver role. The messaging engine performs additional checks where additional access roles

exist. The children of the topic *sports* inherit the receiver role from the parent topic, and from the virtual root. The *lawn* topic also has its has its own receiver role.

You can define access roles for a topic before the topic is created at runtime. Note that the topic can inherit roles from its parent unless you explicitly disable topic inheritance.

Subscription authorization

When a client application creates a subscription for a selected topic, the messaging engine checks that the client has authority in the receiver role to receive messages published to the selected topic. A subscription can be for a single topic, or by using wild cards in the topic specification, for multiple topics. In either case, when a message arrives for the subscription, the messaging engine checks that the client application has authority in the receiver role for the selected topic. When a client application creates a subscription for a single topic, the messaging engine checks that the client has authority in the creator role to create a subscription.

A *non-durable* subscription exists for the duration of the subscriber session. This means that the subscription ceases to exist when the subscriber session ends. A *durable* subscription continues to exist after the subscriber session ends. This means that the subscription continues to collect messages on the topic or topics when the subscriber is not connected to the bus. The subscriber can restart the subscription, and collect the messages. Only the client application that created the durable subscription can restart it. This allows the messaging engine to check for access authority for messages that are collected on the topic when the subscription is inactive.

Access control for multiple buses

The messaging engine on a service integration bus uses role-based authorizations to ensure that local buses can exchange messages securely with foreign service integration buses, and with WebSphere MQ.

Service integration authority is role-based. By assigning a group of user identities in the user repository to one or more access roles for a destination on a local bus, you can control who has authority to access, and undertake operations, on that bus destination. The messaging engine uses the role assignments at runtime to determine which operations group members can undertake on the destination. If a client application needs to exchange messages with another bus, you need to assign the identity of the client application to the sender and receiver roles for the remote (foreign) bus. When the client application sends a message from the local bus destination to the foreign bus destination, the messaging engine performs a two-stage check on the authority of the identity of the client application:

First stage authorization check

When the client application sends a message from the local destination, the messaging engine checks that the identity of the client application has authority in the sender role for the foreign bus destination.

Second stage authorization check

When the foreign bus destination receives the message, the messaging engine checks that the message identity (which is initially set to the identity of the sending client application) has authority in the sender role for the foreign bus destination.

Messages are sent to a foreign bus destination by using either a foreign bus destination proxy definition, or a foreign bus definition. The definition contains the authority that determines whether the sender is authorized to send messages to the foreign bus. Foreign destination definitions enable you to grant authority on a destination by destination basis. If a foreign destination definition does not exist for a specific destination, the messaging engine uses the default foreign bus definition.

It is important to understand that it is the role assignments for the foreign bus, or foreign bus destination, that control whether a client application can send messages to the foreign bus. When the foreign bus receives a message, the messaging engine on the foreign bus uses the foreign bus role assignments to check whether the message can proceed to the foreign bus destination.

For the second stage authorization check, the messaging engine uses the identity that is stored in the message. This is initially set to the identity of the sending client application, but it might be superseded on the foreign bus destination by values specified by the Inbound user ID or Outbound user ID properties. In this case, the messaging engine uses the Inbound or Outbound user ID to undertake its authorization checks on the foreign bus, not the identity of the original sending client application.

Important: If users or groups have the bus connector role for a foreign bus, you must take special care when assigning access roles. In particular, you must check for the following to ensure message delivery between an MQ Link and foreign buses:

- The sending client application must have authority in the sender role for the appropriate foreign bus destinations.
- The foreign bus destinations must exist.

If message delivery is interrupted, it is difficult to diagnose and resolve the problem.

The checks on Inbound and Outbound user IDs also apply when messages are routed through multiple buses, and when messages are sent to a WebSphere MQ network.

Tip: You specify Inbound and Outbound user IDs when you create a foreign bus connection or maintain the routing properties for the link to a foreign bus.

If secure buses are linked, the link between them should be secure. To protect data transmitted along the virtual link between buses by using SSL, you must define the required transport chains and then reference the transport chain name.

Message security in a service integration bus

The transport policy for a service integration bus controls which transport mechanisms a remote client application can use to connect to the bus.

You can configure one of the following transport policies for a bus, providing the bus members are at WebSphere Application Server Version 6.1 or later:

All defined transport channel chains
Connecting client applications can use any transport channel chain, including unsecured ports. This is the default policy when you create a new bus with security disabled.

Transport channel chains that are protected by SSL
Connecting client applications can only use transport chains that use the Secure Sockets Layer channel. This is the default policy when you create a new bus with security enabled.

Transport channel chains in the list of permitted transports
Connecting client applications can only use the transport channel chains in a list of specific transports. This provides the highest level of control because the bus allows access only to the permitted transports.

You can configure the transport policy for the bus by using `wsadmin` commands, or the administrative console. The transport policy is independent of the bus security configuration, so you can configure a transport policy for a bus when security is disabled. Note that by default, if a newly created bus is not secured, a remote client application can use any transport channel chain to access the bus. If a newly created bus is secured, by default a remote client application can only use SSL protected channel chains to access the bus. If you want to control exactly which transport channel chains are available for use, configure the permitted transports policy.

The permitted transport policy provides the following benefits:

- You do not have to disable transport channel chains to prevent remote client applications from using them to connect to the bus.
- You do not have to disable transport channel chains before adding a new server as a bus member.
- Buses that have different transport channel chain requirements can share the same server.

If the permitted transports policy is in use but an inter-bus communications protocol has not been specified, the InboundSecureMessaging port is used instead of the InboundBasicMessaging port. You must ensure that you add the InboundSecureMessaging port to the list of permitted transports. You can override the default by configuring an inter-bus communication protocol for the bus.

Mediations

A mediation is a Java program that extends the messaging capabilities of WebSphere Application Server. Mediations can be used to simplify connecting systems, services, applications, or components that use messaging.

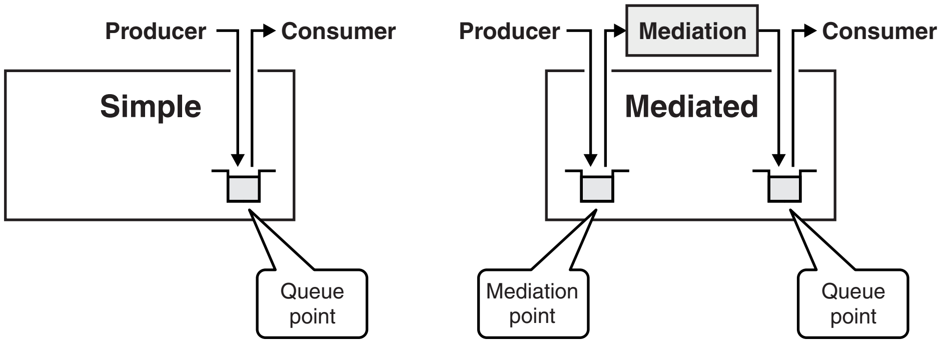
Mediations are used to process in flight messages. The type of processing a mediation can undertake includes:

- Transforming a message from one format into another.
- Routing messages to one or more additional target destinations.
- Adding data to a message from a data source.
- Controlling message delivery based on some conditional logic in the mediation.

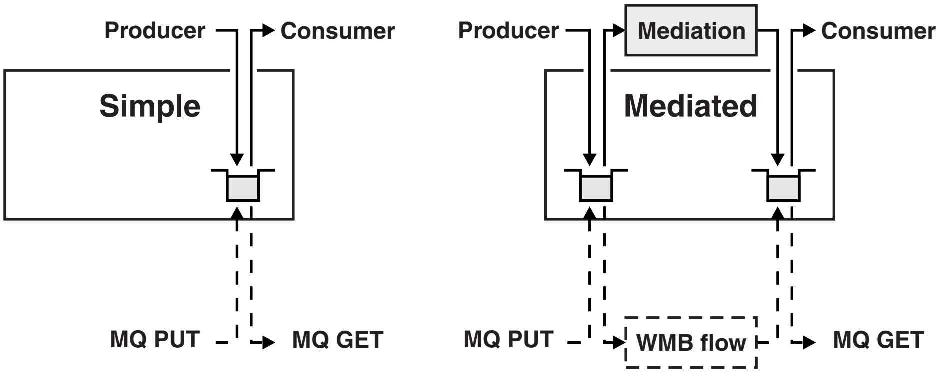
You can use a mediation to process messages as an alternative to using a message-driven bean (MDB). A mediation has two advantages:

- It preserves the message identity. If an MDB re-sends a message after processing its body, it sends a new message with a new message ID and message properties. By preserving the message identity, using a mediation makes it easier to track messages.
- It is independent of the messaging technology. The mediation programming model provides a Service Data Objects (SDO) Version 1 interface to all messages and a common API for accessing properties and metadata.

When a message arrives at the mediation point, the mediation consumes the message and either transforms, subsets, aggregates or disaggregates the message. The message is then either forwarded to another destination or returned to the same destination, in which case, the message goes to the queue point where it can be consumed by the messaging application. This is shown in the following figure:



You can configure a destination so the mediation point or the queue point, or both are WebSphere MQ queues. If both are Websphere MQ queues then a WebSphereMQ application can act as an external mediation as shown in the following figure:



WebSphere Application Server provides a mediation framework runtime that enables you to mediate messages. IBM Rational Application Developer and the assembly tools provide the tools needed to develop, assemble, test and deploy mediations.

You can mediate any type of destination in the service integration bus: inbound or outbound services, queues, and topic spaces. When you mediate a destination it is split into two parts called pre-mediated and post-mediated. The mediation receives messages from the pre-mediated part. Providing the messages are not redirected to another destination or discarded by the mediation, the mediation places messages on the post-mediated part. Messages on the post-mediated part are delivered to a message consumer. Splitting a destination into two parts allows asynchronous mediation of messages.

At deployment, the administrator can choose to have your mediation operate within a global unit of work to ensure transactional integrity, or to support concurrency if throughput of messages at a destination is important.

After deployment, the administrator configures your mediation for use at runtime using the WebSphere Application Server administrative console. The mediation is configured for use at a specific destination. The physical location is called a mediation point. The message processing provided by your mediation is started when the mediation point receives a message from the messaging runtime environment. The mediation operates on the message, for example transforming it, or forwarding it to other destinations.

Mediation handlers and mediation handler lists

Mediations are specified as a simple sequential list of mediation handlers. You must assemble and deploy the mediation handler list into an Enterprise Applications Archive and install it in WebSphere Application Server. Once the mediation is associated with a destination, it processes messages arriving at that destination. A mediation handler is the Java class that processes the messages.

Examples of message processing performed by a mediation handler include the following:

- Transforming a message into a different format.
- Routing messages to other destinations.
- Adding data to a message from a data source.
- Modifying properties of the message.

The mediation handler class implements the Java interface `com.ibm.websphere.sib.mediation.handler.MediationHandler`. You assemble the mediation handler class into an Enterprise Applications Archive (EAR) file then deploy the mediation handler application in a mediation handler list by using an assembly tool, for example IBM Rational Application Developer.

You can assign one or more mediation handlers to a mediation handler list to define a set of operations to apply to each message. When you assign a mediation handler to a mediation handler list, you assign a sequence number to the mediation handler. The sequence number is used to determine the specific sequence in which the mediations in the mediation handler lists are invoked.

You configure and create a mediation in the administrative console, attaching it to a destination. By default the mediation handler list has the same name as the mediation handler, but you can specify a different name if required.

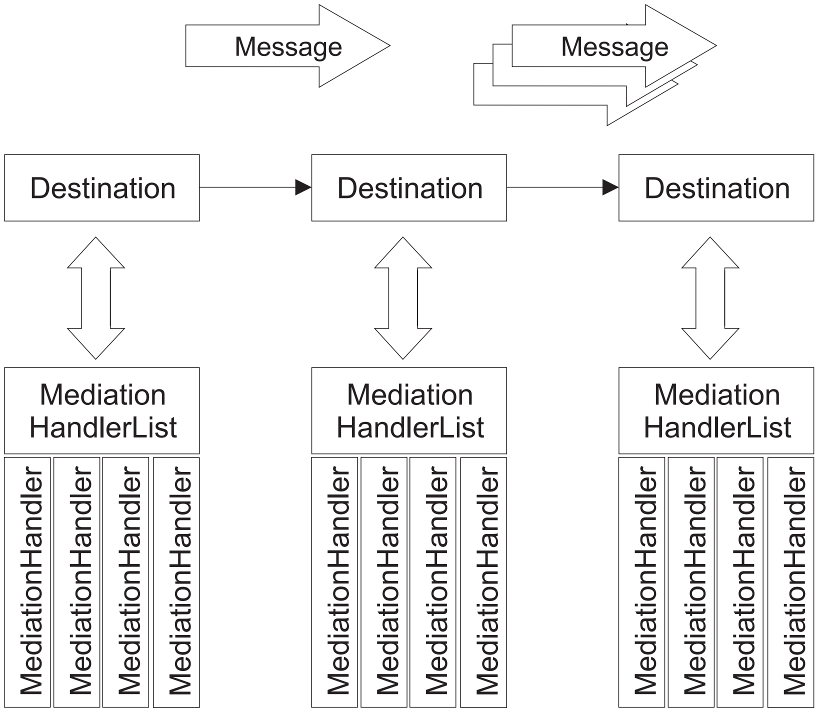


Figure 1 shows how a mediation is composed of a simple sequential list of mediation handlers. The result of the operation of each mediation handler in the list determines whether the next mediation handler in the list is called, or if the message is routed to the next destination.

Figure 87. Mediation handler lists

In most cases, you will assign only one mediation handler to a mediation handler list. The tooling used to deploy the mediation handler into an EAR provides a basic configuration option for automatically adding the mediation handler to its own mediation handler list.

The behavior of a mediation handler can be controlled by setting properties at various stages of mediation development. You can set these properties when you write the Java class, assemble and deploy the mediation handler, or at runtime when the mediation is created and installed.

Transactionality in mediations

You can configure a mediation handler to run within a global transaction.

A global transaction is required when:

- Mediating and routing messages must be coordinated into a single transaction.
- Several mediation handlers in a mediation handler list must be coordinated into a single transaction.

Setting the **Global transaction** property ensures transactional integrity between a mediation that accesses the resources owned by other resource managers, and the messaging engine.

A global transaction encompasses all the mediation operations that are run within the bus for the duration of the mediation. The global transaction ends when the mediation completes its processing.

Whether the **Global transaction** property is set to `True` or `False`, a mediation either performs all its operations on a message, or none of them.

If a mediation transaction rolls back, all transactional changes also roll back. When the transaction rolls back, the mediated message remains on the pre-mediated part of the bus destination and becomes eligible to be mediated again. The re-delivery count assigned to a message increments each time a mediation transaction rolls back. If the re-delivery count exceeds the limit configured for the bus destination, the message is sent to the exception destination.

You specify the transactional context of a mediation handler by setting the **Global transaction** property to `True` by using the administrative console. The default value is `False`, and a global transaction is not started. You can also configure individual messaging operations to run outside the global unit of work.

Performance tuning for mediations

You can set the property **`sib:SkipWellFormedCheck`** in the administrative console to improve the performance of a mediation. Before you set a property to tune a mediation, you should consider the behavior of the mediation, to prevent the modification or loss of messages.

The mediation must either not modify the message, or it must ensure that the message is *well formed* post-mediation. A well formed message has message property values that can be serialized, and a message datagraph that conforms to the message format.

If you set a tuning property for a mediation that does not conform to these rules, the following events might occur:

- Messages disappear either when moving between messaging engines or when they are saved in the message store.
- Modified messages, instead of the original messages, arrive at the exception destination.
- Messages are modified while a mediation or a consumer application is reading the message content.

Tip: When you use an enterprise bean as a message producer, the topic or queue connection factory pool size acts as a throttle that controls the rate at which an enterprise bean can produce messages. There is no one specific value that is suitable for all circumstances, therefore you must tune this parameter for your given application and hardware combination.

Performance monitoring for mediations

Mediation performance monitoring is provided by the WebSphere Application Server Performance Monitoring Infrastructure (PMI).

When PMI is enabled, two pieces of statistical information are updated continuously for each mediated destination, as described in the following table:

Performance statistic	Description
Mediation Time	The time taken to perform the mediation.
Messages Mediated	A count of the number of messages mediated at this destination.

Mediated destination level statistics can be aggregated by mediation. An additional statistic is updated on request for each mediation, as described in the following table:

Performance statistic	Description
Thread Allocated	The number of threads in the mediation thread pool performing work for the mediation.

Mediation level statistics can be aggregated by messaging engine, and messaging engine level statistics can be aggregated to provide statistics for the application server.

Concurrent mediations

Several messages might be mediated concurrently, to achieve maximum throughput for a selected mediated destination.

You can specify that a mediation operates concurrently by setting the **Allow Concurrent mediation** property to `True`, by using the administrative console. If message ordering is important, do not allow concurrent mediations. By default (**Allow Concurrent mediation** is set to `False`), a single message is mediated at a time.

Mediation points

A mediation point is a location in a messaging engine at which messages are stored and mediated.

A *mediation point* is a specialized message point. When an administrator associates a mediation with a bus destination, one or more mediation points are created on the bus member, depending on the type of destination. For a mediated queue (used in point-to-point messaging), a mediation point is created for each queue point on the bus member. For a mediated topic space (used in publish/subscribe messaging), a mediation point is created for each publication point on the bus member.

A mediation point has a property called **`Initial State`** that determines the state of the mediation point when the messaging engine starts. The default value is `Started`.

A mediation point also has a runtime property called **`Send Allowed`** that controls how messages are routed. This property overrides the properties set for the mediated destination.

The mediation point properties are described in the following table:

Table 45. Mediation point properties. The first column contains the mediation point property names. The second column provides the property values. The third column includes the comments of the properties.

Property name	Value	Comment
`Send Allowed`	Boolean: `True` or `False`	`True` routes messages to the mediation point. `False` re-routes messages to a mediation point that has **`Send Allowed`** set to `True`. Messages are not routed to the exception destination. Note that if all mediation points are set to `False`, applications cannot send messages to that destination.
`Initial State`	`Started` or `Stopped`	The default value is `Started`. **`Initial State`** specifies the state of the mediation point when the host messaging engine is started for the first time. It is not a runtime control.

At run time, the administrator can control message delivery by starting and stopping mediations at mediation points by using the administrative console. For example, if the administrator starts a mediation at a mediation point, any messages sent to the destination are processed immediately by the mediation. If the administrator subsequently stops the mediation, any further messages that arrive at the destination are stored at the mediation point until the mediation restarts. The runtime state of a mediation point represents a runtime instance of the mediation.

Mediation context information

Mediation context information is used to ensure that messages are processed correctly by parameterizing the mediation handlers, providing configuration information at runtime. For example, the name of a file to write to.

The mediation context information comprises property values that are passed to each mediation at run time. The property values, in conjunction with the information in the message header, contribute to the way in which a message is mediated.

The context properties (name, value and type) are specified for both mediations and bus destinations by using the administrative console. The mediation uses both sets of context properties, but those for a bus destination takes precedence over the context properties for a mediation. For example, if a property with the same name is configured for a destination and a mediation, the property on the destination takes precedence.

When a message is mediated and a mediation handler is invoked, the information specified in the context information is accessed by the getProperty() method and becomes an entry in the MessageContext information for the mediation handler. The documentation for each mediation handler describes the information it expects to find in the MessageContext property.

When a mediation handler is invoked in the context of a mediation handler list, the message context might be passed from one handler to the next in sequence.

Mediations security

When bus security is enabled, authorization permissions are required to ensure that mediations can run, and undertake messaging operations securely on a service integration bus. There are mechanisms for mediations security, and implications for running mediations on a bus that spans multiple security domains.

When bus security is enabled, the messaging engine must be authorized to access the mediation. Authorization is granted by using a mediations authentication alias or an LTPA token, depending on the version of the bus member:

- A WebSphere Application Server Version 7.0 or later bus member uses an LTPA token for messaging engine authentication. If an authentication alias is specified, it is used but a password is not required.
- A WebSphere Application Server Version 6 bus member requires an authentication alias to ensure that the mediation can be called. For more information, see Configuring the bus to access secured mediations.

When an application sends a message to the bus, the identity of the sender application is associated with the message. The message is sent to the next destination in the forward routing path providing the message originator has `Sender` authority for that destination. If a mediation processes the message in some way at the target destination, the identity associated with the message is preserved by default. You can program the mediation to reset the message identity to the identity under which the mediation code runs. For example, if the mediated destination represents the boundary between two security domains, the sender application is not authorized to access the mediated destination. By translating different identities into a single user identity, you can control access between security domains. For more information about programming mediations, see “Mediation programming” on page 605. For more information about using the resetIdentity() method, see SIMediationSession .

When you install a mediation for use when bus security is enabled, you must ensure that the identity that is used by the bus to call the mediation can access the mediation. By default, a mediation is unauthenticated. You can configure it to use the mediations authentication alias by specifying a RunAs role by using the assembly tools. For more information, see Configuring an alternative mediation identity for a mediation handler.

If bus security is enabled, and a mediation is sending messages to a destination, the mediation identity requires authority to access the destination. For more information, see Administering authorization permissions. Any new messages sent by the mediation are sent using the mediation identity.

If administrative security is disabled, an identity is not configured for the mediation. If bus security is enabled, and administrative security is disabled, the mediation is not authenticated to access bus destinations.

Using mediations in multiple security domains

You can run mediations successfully in a bus topology where the members of a bus span multiple security domains. The bus security configuration provides an option, called **addUserServerIdForMediations**, to allow mediations to run under a server identity. In this case, a mediation authentication alias is not required.

Mediations are deployed as applications, and run in the domain used by the application server, not the bus domain. Because the mediation authentication alias applies to the whole bus, if you run a mediation on multiple servers in different domains, you must ensure that the user identity in the mediation authentication alias exists in the configuration for each domain. Alternatively, you can choose to use the server identity option. You can use this option even if multiple domains are not in use.

Mediation application installation

A mediation application is an enterprise application (EAR file) that contains a mediation handler enterprise bean project. Installing the EAR file into WebSphere Application Server (base) makes the mediation handler available for use at a destination.

You must install the EAR file on every server on which you intend the mediation to run, taking into account the following:

- If you intend the mediation to mediate a topic, you must install the mediation on every server that has a messaging engine.
- If you add a new member to the bus after you have mediated a topic, you must install the mediation on the new bus member.
- If you are installing on a queue, you must install the EAR file on the queue point.

Installing a mediation application on a secure server

If you are installing a mediation application for use when WebSphere Application Server security is enabled, you must ensure that the messaging engine can access the mediation. For more information, see Ensuring the message engine can access mediations.

Mediation programming

Using the capabilities of the mediation infrastructure, you can program mediations to customize the way that a service integration bus handles messages. You develop the mediation code within a component called a mediation handler, and add the mediation handler to a handler list, which is an application that is ready to deploy and install. You can connect a number of mediation handlers together in a mediation handler list to create a set of operations to run on a message.

A mediation handler is a Java program framework to which you add the code that operates on a message to perform the mediation function. For example, you can program mediations to process messages in any of the following ways:

- Reformat messages from the format produced by one application to the format required by another
- Route messages based on message content
- Distribute messages to more than one destination

- Augment messages by adding information to a message from another data source
- Transcode messages from one concrete representation to another

The following programming APIs are available for working with messages when you program a mediation:

MediationHandler API
A mediation handler must implement the MediationHandler interface. This interface defines the method that is invoked by the mediation runtime environment.

SIMessage and SIMessageContext APIs
These APIs allow your mediation to operate on the contents of the message.

SIMediationSession API
This API gives your mediation access to a service integration bus so that the mediation can send and receive messages.

You create a handler list by using an assembly tool, for example IBM Rational Application Developer, before deploying the mediation handler application as an Enterprise Archive (EAR file).

A handler list can contain one or many mediation handlers. At run time, each mediation handler in the list is invoked in sequence. Each time a handler returns a value of *True*, the same message context is passed to the next handler. If a handler returns the value *False*, then the context is not passed to the next handler. The message is discarded, and is not delivered to its target destination.

SDO data graphs

Service Data Objects (SDO) is an open standard for enabling applications to handle data from different data sources in a uniform way, as data graphs. SDO data graphs are an important concept for mediation programmers because you can use them to represent different types of message information in a standard way, giving a simple and powerful model for programming mediations.

Using SDO, your applications can uniformly access and manipulate data from diverse data sources including relational databases, XML data sources, Web services, and enterprise information systems.

SDO data graphs are structured collections of data objects. In general, data graphs generated from messages have a tree structure. A mediation retrieves a data graph from a message, transforms the data graph, and reflects the updates to the data graph in the message.

In WebSphere Application Server, data access services connect mediations to data sources, allowing mediations to manipulate an abstract representation of the message, the SIMessage. The SIMessage API provides a method, getDataGraph(), that returns the SDO data graph containing the SIMessage content in a tree representation, or graph of data objects. Each data object represents one or more fields in the message, or it points to other objects.

When a data graph is requested from a message, the appropriate data access service is identified by a format property in the SIMessage. The format string controls which data access service is used to process the message, and can contain additional control information for that data access service. The data access service controls the structure of the message. For more information about the data access services available in WebSphere Application Server, see Mapping of SDO data graphs for web services messages.

The SIMessageContext API provides access to:

- The SIMessage, and its rich set of message manipulation methods
- The SIMediationSession, for Service Integration technologies functions

Data objects hold their data as a set of named properties. Each property has a type that is either an attribute type (for example, int) or a commonly used data type (for example, Date). If the property is a reference, it has the type of another data object. The Data Object API provides a dynamic data API for manipulating these properties, with the following interfaces that relate to instance data:

- The **DataObject** interface provides a set of methods to retrieve and update the contents of a data object. It also provides methods to perform the following actions:
  - Access the container of the data object and the data graph to which the data object belongs
  - Create a new instance of a contained data object
  - Delete a data object from its container

  The DataObject interface also provides the ability to get the type of the data object.
- The **DataGraph** interface is a graph of data objects. The graph consists of a single root data object and all the data objects that can be reached by recursively traversing the containment references of the root data object.

SDO also contains a metadata API for examining the model of a data graph, consisting of Types and Properties:

- A Type has a set of Property objects. SDO Types can be compared with type definitions in other type systems. For example, the SDO view of a Java Class is a Type, with each field in the Class represented by a Property. For XML Schema, a ComplexType is represented by a Type, with a Property for each element or attribute.
- A data object is composed of properties. To access a property, specify the Property object, the name of the property, or the index of the property.

The version of SDO data graphs used by mediations is Version 1. Data graphs provided by the SIMessage and SIDataGraphFactory interfaces can only be provided to other SIMessage objects. Data graphs provided to SIMessage objects can only come from other SIMessage objects or have been constructed using the SIDataGraphFactory.

Coding tips for mediations programming

Programming hints to help you when you are writing mediation code.

- Take care to avoid looping in the Forward Routing Path. For example, if you set a destination in the path that is the same as the current destination, the message will endlessly circle, with the routing path being reset to the current destination each time. The mediation framework does not check for loops in routing paths.
- Avoid the use of static fields where possible. A single mediation may be deployed to process multiple messages concurrently.
- Do not cache values computed from the message context or message contents. Such values might change from message to message. The exception is caching values derived solely from the mediation handler properties for performance purposes.
- Mediation programming is subject to the same restrictions as programming an EJB. For more information about restrictions, see Section 18.1.2 of the EJB 1.1 specification.
- Choose the appropriate level of transactional control for your mediation: for example, a mediation that operates on fields within a message is unlikely to have implications for transactional control. At the other extreme, if your mediation updates database fields, it requires transactional control, and you should alert your administrator to set the UseGlobalTransaction flag in the mediation definition. This flag defaults to a value of *False*.
- Hints that apply specifically to message format:
  - It is good practice to check that your message conforms to the expected format after your mediation function has operated on it. You should use the isWellFormed method in the SIMessage interface to check that all the values of the message properties can be serialized, and that the data graph of the message conforms to the format of the message.
  - Depending on how you want to process the message, you can specify a format that meets your needs rather than accept the natural format. For example, if you want to handle a SOAP message as a byte string, use the getNewDataGraph method in the SIMessage interface and specify a format

of JMS/bytes. getNewDataGraph returns a new SDO data graph containing a copy of the SIMessage payload content in the tree representation specified by the format field, in this example as a byte string.

  – It is good practice to check the message format in the mediation code because a mediation is unlikely to successfully process a message with an unexpected format. Use getFormat method on the SIMessage interface.

- The version of SDO supported by mediations is Version 1 only.
- Due to a restriction in the SDO user interface to the message, message access methods do not have a ‘throws' clause. As a result, an exception thrown by an access method because of a parsing error is an unchecked exception. Your mediation can catch a parsing exception by checking for the exception class SIMessageParseException in the com.ibm.websphere.sib.exception package. Use code similar to the following example:

```
try {
  // Function involving SDO message access
} catch (SIMessageParseException e) {
  // Look at the real cause of the runtime exception, and act on it.
  // It is likely to indicate a parse failure...
  Throwable cause = e.getCause();
}
```

  Note: If a mediation does not catch the SIMessageParseException, the original version of the message is sent to the exception destination.

- When deploying your mediation, give the handler and the handler list memorable and descriptive names.
- Where you deploy a single mediation against a single destination, use exactly the same name for your mediation handler, the mediation handler list and the mediation object in the administrative console.
- For performance reasons, specify selector rules so that the mediation mediates required subsets only of the messages passing through a destination.

Service integration configurations

A service integration configuration can range from a single host running two connected applications to a globally-dispersed set of hundreds or thousands of communicating applications running over the bus.

A service integration configuration is based on one or more service integration buses that provide a managed communication fabric that supports service integration through asynchronous messaging.

A bus is a group of one or more interconnected bus members, each of which is an application server or an application server cluster. Applications connect to a bus at one of the messaging engines associated with its bus members.

A service integration bus provides the following capabilities:

- Any application can exchange messages with any other application by using a destination to which applications send and receive messages.
- An application can produce messages for a destination regardless of which messaging engine the producer uses to connect to the bus.
- An application can consume messages from a destination (whenever that destination is available) regardless of which messaging engine the consumer uses to connect to the bus.
- The service integration bus is the default messaging provider for JMS applications.

Many scenarios only require a simple bus configurations, for example, a single server. If you add multiple servers to a single bus, you increase the number of connection points for applications to use. If you add server clusters as members of a bus, you can increase scalability and achieve high availability. Servers however, do not have to be bus members to connect to a bus. In more complex bus configurations, multiple buses are configured, and can be interconnected to form complex networks.

An enterprise might deploy multiple interconnected buses for organizational reasons. For example, an enterprise with several independent departments might want a separately administered bus in each location.

With bus-enabled Web services you can achieve the following goals:

- Create an inbound service: Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Create an outbound service: Take an externally-hosted Web service, and make it available internally at a bus destination.
- Create a gateway service: Use the Web services gateway to map an existing service, either an inbound or an outbound service, to a new Web service that appears to be provided by the gateway.

You can change the service integration configuration to suit your needs, for example:

- You can add application servers as new bus members. Each new bus member is automatically assigned a messaging engine, with default data source, and a default exception destination. The messaging engines can communicate with, and use resources provided by, all other messaging engines on the bus.
- You can change the configuration of the data source for a messaging engine, for example to use a different JDBC provider.
- You can create new buses and add application servers as members of those buses. Each bus operates as a separate environment, unless connected by a gateway messaging engine.
- You can connect a message-driven bean to consume messages from a destination on a remote cell.

Bus configurations

You can connect buses in different ways depending on your requirements. For example, you can link messaging engines to distribute message workload, and to provide availability if there is a system failure.

A configuration that only has a single messaging engine might be adequate for some applications however, deploying more than one messaging engine and linking them together provides the following advantages:

- Messaging workload is distributed across multiple servers.
- Message processing is positioned close to the application that is using it, and reduces network traffic. For example, if both sending and receiving applications are running in the same server process, it is inefficient to route all the messages that flow between the two applications through a messaging engine running in a remote server.
- Availability is improved in the event of system or link failure. For example, your bus topology can remove a single point of failure, and allow store and forward between two servers.
- Options for scalability improve.
- Firewalls, or other network restrictions that limit the ability of network hosts to connect to a single messaging engine, can be accommodated.
- A bus configuration can contain links toWebSphere MQ networks. This allows messages to flow between applications connected to a WebSphere MQ queue manager and applications attached to a service integration bus.

The application servers that host a messaging engine in the service integration bus are called bus members. A WebSphere MQ server is the WebSphere MQ equivalent of a messaging engine. You can make a WebSphere MQ server a member of a bus, which becomes a messaging engine which is not hosted by an application server.

A bus configuration can include one or more bootstrap members. When an application needs a connection to the bus, it connects to the bootstrap member, which authenticates the request, and then directs the connection request to a suitable bus member. A bootstrap member responds only to bootstrap requests and does not always host a messaging engine.

If a bus configuration uses multiple security domains, you can isolate buses and the applications that use them by configuring the bootstrap members so that only a subset of servers can access a bus.

Single-server bus

The simplest configuration is a bus consisting of a single server. Use this configuration if there is a low volume of message throughput and scalability is not essential.

In a single-server bus, there is one messaging engine. All destinations, such as queues and topic spaces, are assigned to this single messaging engine.

The single server set up has the advantage of simplicity. It aids performance as all messages and application connections are on the same messaging engine which minimizes path length. It also easy to manage as all messages and application connections are on the same messaging engine which minimizes the number of configuration and runtime objects to monitor.

However, having only a single-server configuration has the drawback of limiting scalability and high availability of applications and messaging.

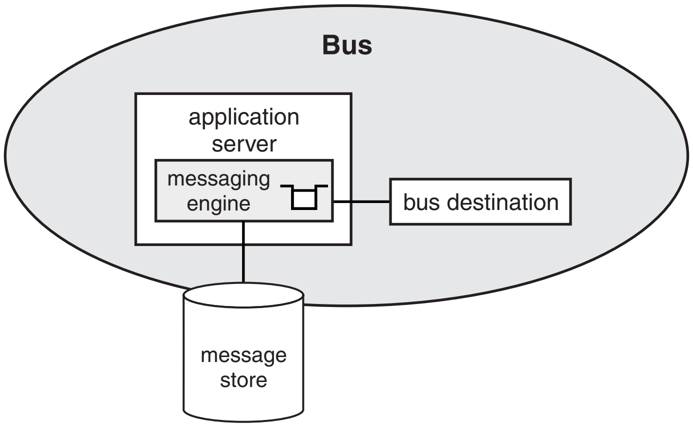


Figure 88. Service integration bus with a single member

An application can connect to the messaging engine, and therefore can connect to and use the bus, in any of the following situations:

- The application runs in another server in the same cell or in the same server, or in a server in a different cell , or in a client container.
- The application uses a client connection to the bus, or a in-process call

The following figure shows possible connections between a messaging engine and an application:

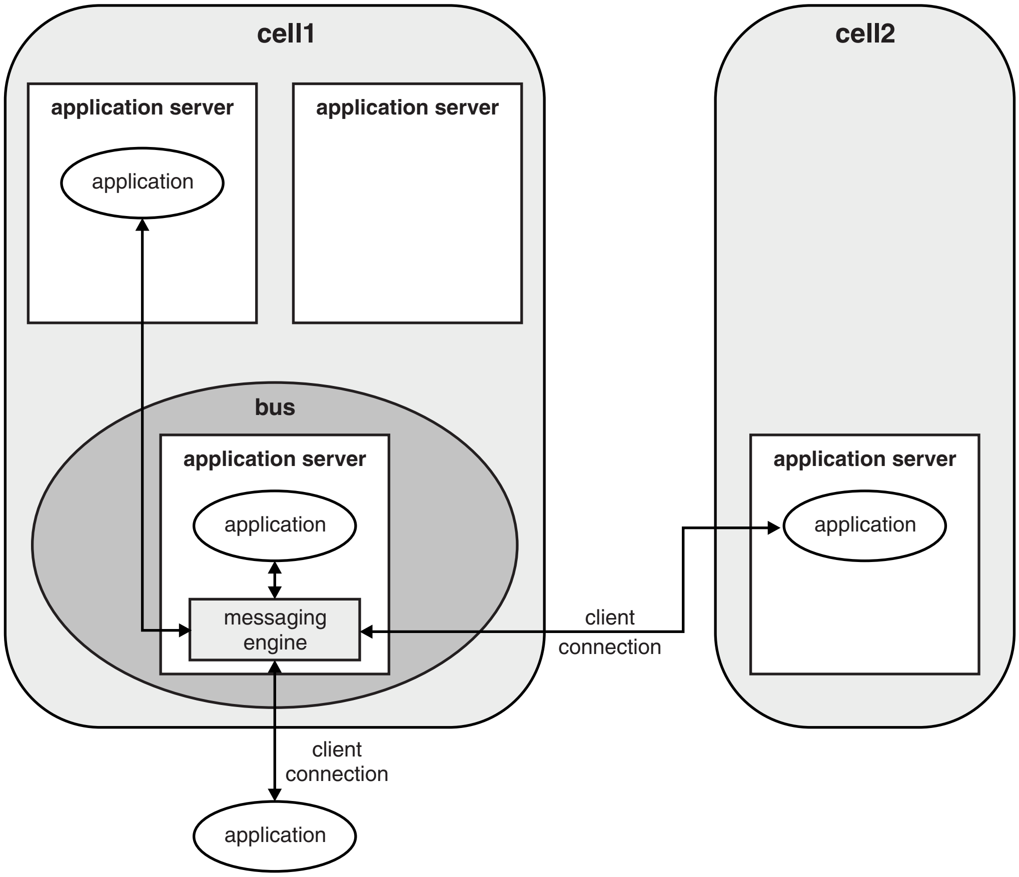


Figure 89. Applications connecting to a messaging engine

Common issues with all bus configurations

There are planning issues and design decisions that apply to all types of service integration bus configuration.

When planning a service integration bus configuration, consider the following points:

- The volume of messages that a bus has to handle. Depending on the volume of messages anticipated, you might have to adjust the high message threshold setting for a bus or messaging engine.
- The transport chain to be used for communication between messaging engines. For more information, see Transport chains.
- Whether bus security is required. When bus security is enabled, access to the bus itself and to all destinations on the bus must be authorized. If you enable bus security, you might also want to define aliases for authenticating messaging engines and mediations accessing the bus. A single version bus does not require an authentication alias. However, if you create a mixed-version bus you must define an inter-engine authentication alias for a WebSphere Application Server Version 6 or Version 6.1 bus member, to enable it to establish trust with the other bus members of later versions.
- You must choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after the bus is created, which means that you can only interoperate with WebSphere MQ in the future if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.

- When you name your buses, you must ensure that the names are unique because you cannot connect two buses with the same name. For example, you cannot connect two buses with the same name in any of the following ways:
  - By creating an inter-bus link between two buses with the same name.
  - By attempting to connect to a remote bus from an application running in a remote cell where a bus with the same name is defined.
  - By creating a core group bridge between two cells containing buses with the same name.

Destinations

You must decide on the number and type of destinations, mediations, destination routing paths, and qualities of service for the destination for your configuration. For point-to-point messaging you define bus destinations as queues, whereas for publish/subscribe messaging you define bus destinations as topic spaces.

For point-to-point messaging only, you select one bus member as the assigned bus member that is to hold messages for the queue. This action automatically defines a queue point for each messaging engine in the assigned bus member.

You can also define alias destinations to provide a level of indirection between applications and the underlying target bus destinations. Applications interact with the alias destination, so you can change the target bus destination without changing the application.

You should decide how you want to use the bus destinations as you can configure a bus destination to prevent producers sending messages to the destination, or consumers receiving messages from the destination.

Message persistence

The reliability quality of service for messages on a destination has implications for performance and the amount of space required for a message store. Higher levels of reliability impact performance and increase the space a message store requires, because fewer messages are discarded.

When planning a message store configuration, remember that each messaging engine has a single message store, which can be either a file store or a data store. See “Relative advantages of a file store and a data store” on page 574. Remember that larger messages increase the space that a message store requires.

If you use a data store, the default database system for the data store is Apache Derby Version 10.3. However, you might want to use a different system, such as DB2. You can select different data store configurations depending on your requirements; for more information see “Configuration planning for a messaging engine to use a data store” on page 582.

Application environment

An application attaches as a client to a messaging engine on the bus, either by an in-process call or across a network by using a remote client. A remote client can run in either the Java EE application client environment or the Java EE application server environment. Various transport chains can be used.

Application connections

The way that a messaging engine is selected, and the mechanism that an application uses to reach it, is configured on a JMS connection factory. You need to decide which messaging engines the applications should connect to and on the transport chain to be used. For more information about connection factories, see Configuring resources for the default messaging provider and on transport options, see Transport chains.

WebSphere MQ client links

WebSphere MQ client links allow JMS clients developed for WebSphere Application Server Version 5.1 to use messaging resources on the bus. WebSphere Application Server Version 5.1 uses a WebSphere MQ queue manager as its JMS provider so that WebSphere Application Server

Version 5.1 clients connect using the MQ link protocols. A WebSphere MQ client link, in service integration, provides an attachment capability that these clients can use.

Transaction logs
Plan where transaction logs will be placed. See the topic about transactional high availability in the related links.

Configurations that include WebSphere MQ

There are additional points to consider when planning a bus configuration that includes WebSphere MQ.

You might want to define alias destinations to map bus and destination names to targets where the bus name, or the destination name (identifier), or both, are different. You can also use alias destinations to manage situations where the difference in the name length that is allowed for a bus destination in WebSphere Application Server and the name length that is allowed for a WebSphere MQ queue, might cause a problem.

You might want to define foreign destinations so that you can override the messaging defaults or security settings for specific destinations on a foreign bus. If you do not define either a foreign destination or an alias destination, the destination defaults for the foreign bus will be used.

Remember you can have more than one messaging engine with a WebSphere MQ link in a service integration bus. You can also have more than one WebSphere MQ link on a single messaging engine. There are many possible configurations, for example:

- One WebSphere MQ link engine with only a sender channel and another WebSphere MQ link engine with only a receiver channel.
- One WebSphere MQ link to communicate with a WebSphere MQ queue manager or queue-sharing group (known as a “gateway queue manager”) in the WebSphere MQ network.

Although you can have more than one WebSphere MQ link on a single messaging engine, each WebSphere MQ link must connect to a different gateway queue manager.

Multiple bus member bus

Multiple bus members should be considered for your bus configuration where messaging traffic is high and can be suitably divided, for example, per application or queue.

The following figure shows a configuration which is very similar at runtime to any single messaging engine configuration when application messaging is self contained.

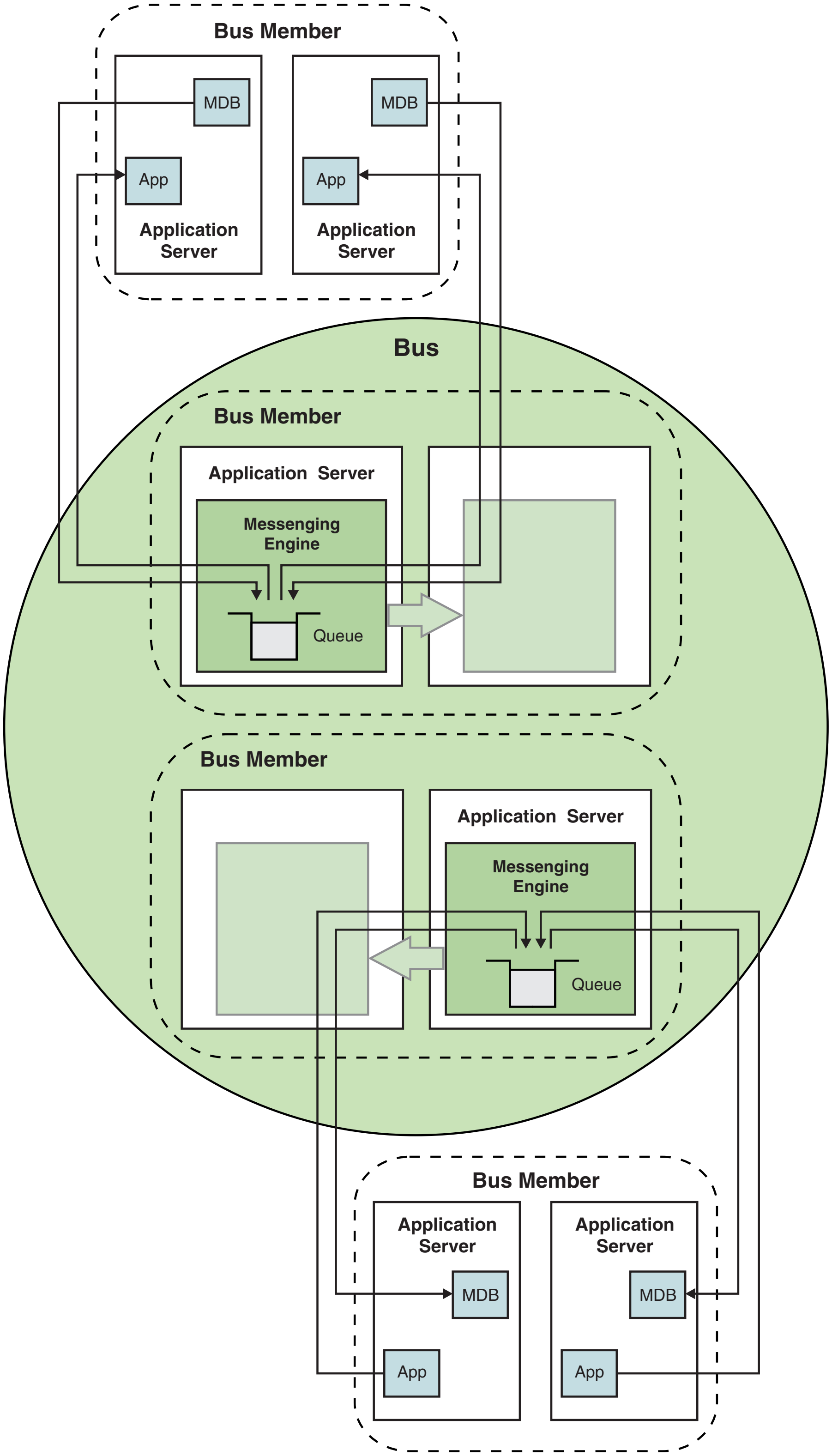


Figure 90. Multiple bus member bus configuration

However once applications on different servers start to communicate with each other using messages, the runtime starts to change.

Careful planning is important for a multiple bus member bus configuration because messaging applications will randomly connect to any available messaging engine in the bus. The only default behavior that overrides this is when an application connects to a bus that has an messaging engine running in the same server. This can result in messages taking inefficient paths to and from the applications to queues or subscriptions. This affects manageability and serviceability of the system due to the unpredictable nature of connections and variable message routing.

The general rule is to connect directly to the bus member that owns the queue. Always target connections and activation specifications. If you have to choose between a producing application or a consuming application, connect the consumers directly to the queue point's messaging engine and allow the producer to store and forward.

Interconnected bus configurations

There are specific issues that you must take into account when you are planning an interconnected service integration bus configuration.

When you are naming service integration buses, bear in mind that bus names must be unique.

You must decide what your buses are to be linked to. You can link the buses either through a direct service integration bus link, or through an indirect link. An indirect link can include one or more intermediate buses. For more information, see “Direct and indirect routing between service integration buses” on page 629.

You must decide which messaging engines to use as gateways. Remember that a gateway messaging engine connects to the gateway messaging engine of another bus through a service integration link.

Carefully plan how you distribute destinations on different messaging engines in each bus. You might want to define alias destinations that make a destination available by a different name, either on the same bus, or on a foreign bus. You could define foreign destinations which allow applications on one bus to directly access a destination on a foreign bus. If you do not define foreign destinations, you can configure destination defaults to be used. You can combine alias and foreign destinations for further flexibility in your topology.

Use destination defaults in the following scenarios:

- You have a development environment and want things to work quickly.
- You have an application in which destination names are received at run time in message body or headers.

Use foreign destinations in the following scenarios:

- You want an environment in which everything is statically defined.
- You want to override destination defaults for a particular (foreign) destination, for example quality of service settings.

Use an alias destination in the following scenarios:

- You want to refer to a destination by a different name. You might want to use a different name if you want to be able to control which users have different access to the same destination in a foreign bus. In this case you might need to use foreign bus destinations definitions or alias bus destination definitions, or both.
- You want multiple names for the same destination.

There is a security consideration that arises from having a mixed-version bus. In a mixed-version bus, you must define an inter-engine authentication alias for aWebSphere Application Server Version 6 or Version 6.1 bus member, to allow it to establish trust with the other bus members of later versions. In the case of a single version bus, you do not need to define an inter-engine authentication alias to ensure the secure operation of the bus.

If buses in different organizations are connected, you must decide whether to secure connections to a foreign bus with a user ID and password, and optionally with SSL authentication.

Interconnected buses

A service integration bus topology can contain many interconnected service integration buses to form a large messaging network. The bus that an application connects to is called its local bus. There can be connections from that local bus to other service integration buses, which are called foreign buses. Buses can also be linked to WebSphere MQ resources, for example WebSphere MQ queue managers. WebSphere MQ resources are also regarded as foreign buses.

A bus must be contained in a single cell; that is, a bus cannot span multiple cells. However, a cell can contain more than one bus. In this situation, each bus in the cell is foreign to each other bus in the cell. You can connect buses together in a cell, or between different cells.

The following scenarios are examples of situations when you might connect service integration buses in an organization:

- You can deliberately separate the messaging infrastructure to aid management.
- You can restrict access to certain messaging resources within a single WebSphere Application Server cell, because a cell can contain multiple service integration buses.
- You can span multiple administrative cells, by connecting a service integration bus in one cell to a service integration bus in another cell.

When buses are connected, applications can send messages to applications on other buses, and use resources provided on other buses. Published messages can span multiple buses where the connections between the buses are configured to allow it.

To create a connection between two buses, the administrator of the local bus configures a *foreign bus connection* that represents the second bus, and that is associated with the local bus. The foreign bus connection contains a routing definition, or virtual link. A physical link, called a *service integration bus link*, is created automatically. The link is from a messaging engine in the local bus to a messaging engine in the foreign bus, and these two messaging engines are known as *gateway messaging engines*. The administrator of the second bus also configures a foreign bus connection to represent the first bus, as a property of the second bus.

To create a link between a bus and a WebSphere MQ queue manager, the administrator of the local bus configures a foreign bus connection that represents the WebSphere MQ queue manager, as a property of the local bus. The foreign bus connection contains a routing definition, or virtual link. A physical link, called a *WebSphere MQ link*, is created automatically. The link is from a messaging engine in the local bus to a queue manager or queue sharing group in the foreign bus. The messaging engine is known as a gateway messaging engine, and the queue manager or queue sharing group is known as the gateway queue manager.

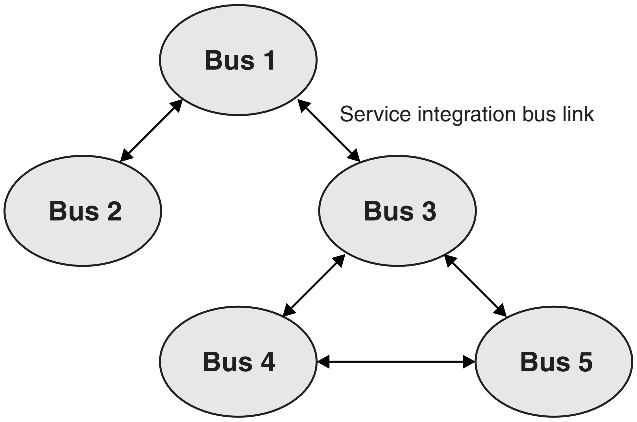


Figure 91. Service integration buses are connected through service integration bus links

Routing between buses

The route between two buses can be indirect, passing through one or more intermediate foreign buses. In Figure 1, Bus 1 is connected to Bus 5 indirectly. For more information about direct and indirect routing between service integration buses, refer to the subtopics.

For more information about foreign buses, see “Foreign buses” on page 618. For conceptual overviews of point-to-point and publish/subscribe messaging, see “Point-to-point messaging across multiple buses” on page 622 and “Publish/subscribe messaging across multiple buses” on page 624.

Security when connecting buses

A multiple bus topology has the following security requirements:

- You must protect the integrity and confidentiality of the data that is transported between the buses. You can protect the communications links by using a Secure Sockets Layer (SSL). For more information, see Protecting messages transmitted between buses.
- You must establish trust between two buses. Trust between messaging engines at WebSphere Application Server Version 7.0 or later is established by using a Lightweight Third Party Authentication (LTPA) token, and no further configuration is required.

  If the bus has a WebSphere Application Server Version 6 bus member (that is, a mixed-version bus), trust is established using an inter-engine authentication alias. The inter-engine authentication alias is configured when you add a member to a bus or with the bus security settings. The identity is passed to the remote bus where the identity is authenticated, then checked to see if it matches the configured inter-engine authentication alias on the other bus.
- You need the definition of relevant authorizations to allow messages to travel between the buses. There are two phases to authorization when communicating with a foreign bus:
  1. The user that is connected to the local bus has to be explicitly granted access to send messages to the foreign destination. Failure at this level is reported back to the client.
  2. The foreign bus must be configured to accept the incoming message onto the target destination.

For more information about security, see “Service integration security” on page 583 and Securing access to a foreign bus.

Connecting buses in different cells

To connect a local bus to a foreign bus in a different cell from the local bus, you need to provide a value for one or more bootstrap endpoints, that is, the host, port location, and transport chain for the messaging engine on the foreign bus that the local service integration bus connects to.

Connecting buses with cluster bus members

To connect a local bus to a foreign bus in a different cell from the local bus when the remote messaging engine is in a cluster, you must change the value for the bootstrap endpoints. This value must list all the bootstrap endpoints that the cluster uses to allow access to the gateway messaging engine in the cluster.

For more information, see the steps relating to setting bootstrap endpoints in Configuring a connection to a non-default bootstrap server.

Foreign buses:

You can configure a service integration bus to connect to, and exchange messages with, other messaging networks. To do this, you configure a foreign bus connection, which represents either another service integration bus, or a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group, that the existing service integration bus can exchange messages with. In this way, you can extend the network of buses that can exchange messages.

When an application connects to a service integration bus, that bus is its local bus. A foreign bus is any other bus that has a link to the local bus. When the foreign bus is a service integration bus, it can be in the same cell as the local bus, or in a different cell.

To exchange messages between two buses, you configure a foreign bus connection from the local bus to the second bus. The foreign bus connection is associated with the local bus, and identifies the second bus as a foreign bus.

If the second bus is another service integration bus, you then configure a foreign bus connection from the second bus to the first bus. The foreign bus connection is associated with the second bus, and identifies the first bus as a foreign bus relative to the second bus. If the second bus is in a different cell from the first bus, you use the administrative console for the second cell to configure this foreign bus connection.

A foreign bus connection can be direct or indirect. For a direct foreign bus connection, messages route directly through a link between the local bus and the foreign bus. For an indirect foreign bus connection, messages route indirectly through one or more intermediate buses.

A foreign bus connection contains a routing definition, also known as a virtual link, which indicates the type of physical link:

- A *service integration bus link* specifies a link from a messaging engine in the local bus to a messaging engine in a foreign bus.
- A *WebSphere MQ link* specifies a link from a messaging engine in the local bus to a WebSphere MQ gateway queue manager. To the local bus, the linked WebSphere MQ network appears as a foreign bus.

In the following figure, for an application that is connected to Bus 1, messages that are routed to Bus 2 use a direct foreign bus connection. Messages that are routed to Bus 3 use an indirect foreign bus connection and use Bus 2 as an intermediate bus.

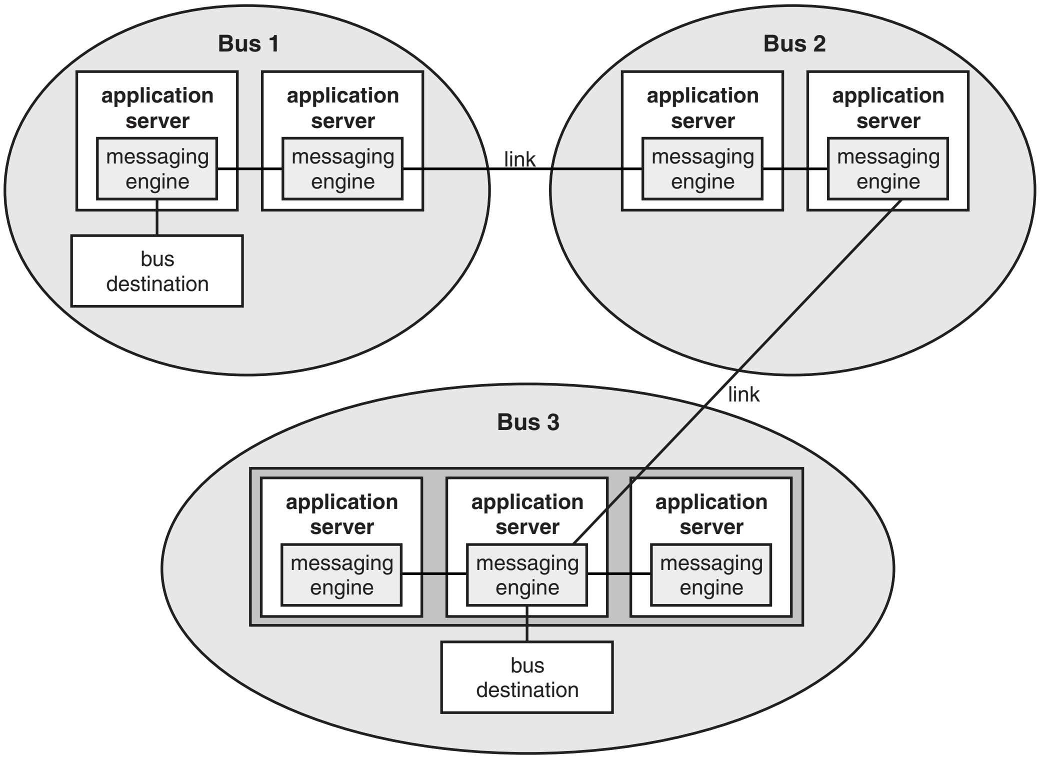


Figure 92. Linking service integration buses.

When you configure a foreign bus connection, the service integration bus link or WebSphere MQ link, as appropriate, is created automatically. If required, you can view or amend this link.

The following restrictions apply when you configure a foreign bus connection:

- The name of the foreign bus in the foreign bus connection must match the name of the existing service integration bus that it represents.
- For a direct foreign bus connection, the name of each bus must be unique.
- You must specify the same user ID for both foreign bus connections on each side of a service integration bus link, for the following reasons. Consider two messaging engines, A and B, connected by a service integration bus link:
  - Messaging engine A presents the user ID and its password to messaging engine B, so that messaging engine B can authenticate messaging engine A.
  - Messaging engine A uses the user ID to authorize messaging engine B.
- After you configure a foreign bus connection, you must not change the name of the service integration bus that the foreign bus connection represents.

When you configure a bus, you can select the **Configuration reload enabled** option so that if the configuration properties of any foreign bus connections are changed later, the changes are updated automatically. The time when these changes take effect varies, depending on the properties that are changed:

- Foreign bus connection properties change immediately.
- WebSphere MQ link properties change on channel restart, except Description (immediately), and Initial State (on messaging engine restart).
- MQ sender channel properties change on channel restart, except Initial State (on messaging engine restart or sender channel creation).

- MQ receiver channel properties on channel restart, except Initial State (on messaging engine restart or receiver channel creation).
- Publish/subscribe broker profile (0 to n) properties change immediately.
- Service integration bus link properties change on link restart, except Description (immediately), and Initial State (on messaging engine restart or link creation).

You can define an explicit destination on a foreign bus that an application can send messages to. You can also configure default properties for use by messages that are sent to destinations on a foreign bus when there is no explicit foreign destination definition, and the application does not explicitly provide values for the properties. An application cannot receive messages from a foreign destination; it can only consume messages from a destination on the bus to which it is connected.

Messages flowing to or from a foreign bus that cannot be processed successfully are rerouted to the system exception destination of the messaging engine that owns the link to the foreign bus, possibly disrupting message order. Common reasons for rerouting messages to the exception destination are that the target destination is unknown by the foreign bus, or that the foreign bus has not granted the sending bus access to the target destination.

An application subscribing to a local topic space can receive messages published to a topic on a foreign bus. To allow publish/subscribe messaging between buses, you must map topic space names on a local bus to topic space names on a foreign bus.

A topic space mapping allows subscribers on the local topic space to receive messages published in the foreign topic space. For publications to flow from the local topic space to the foreign bus, an equivalent topic space mapping is required by the foreign bus.

You administer topic space mappings when you create a foreign bus connection, or through the routing properties for a foreign bus connection. Topic space names for the local bus are mapped to topic space names that are defined on the foreign bus. It is common for these two names to match. Note that mapping two topic spaces implies that the topics in them are the same.

Message flow between service integration buses:

An application connects to a bus, which is its local bus, and can exchange messages with other applications that connect to the same bus. To exchange messages with applications that connect to a different bus, that is, a foreign bus, you need a service integration bus link that connects the local bus to the foreign bus.

Applications that are connected to the local bus send messages to a destination on a foreign bus. The messaging engine on the local bus queues the messages on its link transmitter queue. For applications that use point-to-point messaging, there is one link transmitter queue and one link transmitter for each messaging engine in the sending bus. For applications that use publish/subscribe messaging, there is one link transmitter queue and one link transmitter for each topic space destination that is mapped to a topic space destination in the foreign bus.

Each link transmitter queue has a corresponding link receiver queue on the gateway messaging engine on the foreign bus. Each link receiver queue is served by a link receiver.

The link transmitter sends the messages over the service integration bus link to link receiver queues. Link receivers remove the messages from the link receiver queues and place them on the target remote queue points, which are on the gateway messaging engine on the foreign bus. Both point-to-point and publish/subscribe messaging in service integration use link transmitter queues.

The following figure shows an example of the message flow from Service integration bus 1 to a foreign destination on Service integration bus 2 over a service integration bus link. One application is connected to the messaging engine ME1, and another application is connected to messaging engine ME2. The applications produce messages to send to Q1 on the foreign bus. Messages are queued on the link transmitter queue on each messaging engine, then transmitted through the gateway messaging engine and the service integration bus link to the link receiver queues on the gateway messaging engine in the foreign bus. From here, the messages are placed on to the target destination Q1. The target queue Q1 is on the messaging engine ME5. The link receiver in the gateway messaging engine ME4Gateway routes the messages to ME5 by using a remote queue point.

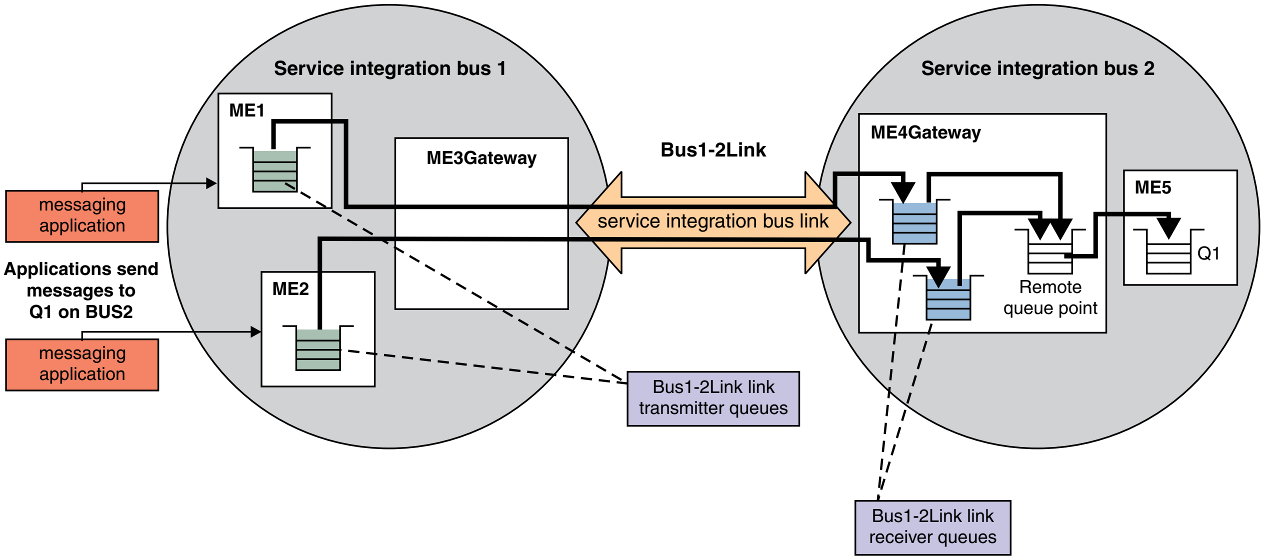

Service integration bus 1
ME1
ME3Gateway
messaging application
Applications send messages to Q1 on BUS2
ME2
messaging application
Bus1-2Link
service integration bus link
Bus1-2Link link transmitter queues
Service integration bus 2
ME4Gateway
ME5
Q1
Remote queue point
Bus1-2Link link receiver queues

Figure 93. Message flow between two service integration buses

Message flow between a service integration bus and a WebSphere MQ network:

An application connects to a bus, which is its local bus, and can exchange messages with other applications that connect to the same bus. To exchange messages with applications that connect to a WebSphere MQ network, you need a WebSphere MQ link that connects the local bus to a foreign bus that represents a WebSphere MQ network.

Applications that send a message to a queue in a WebSphere MQ queue manager or queue-sharing group can do so directly by configuring a WebSphere MQ server definition, or indirectly by using a WebSphere MQ link. This topic describes the message flow for a WebSphere MQ link.

With a WebSphere MQ link, there is a gateway messaging engine on the service integration bus and a gateway queue manager on the WebSphere MQ network.

Applications that are connected to the local bus send messages to a destination on a foreign bus. The messaging engine that the sending application is connected to on the local bus queues the messages on its link transmitter queue. Service integration flows the messages from the link transmitter queue to the corresponding known link transmitter queue in the gateway messaging engine. Messages then flow to a single sender channel transmitter queue, ready for transmission across the WebSphere MQ link.

The sender channel transmitter transmits messages over the WebSphere MQ link to a gateway queue manager or (for WebSphere MQ for z/OS only) a queue-sharing group on the remote WebSphere MQ network.

The WebSphere MQ network appears as a foreign bus to the service integration bus and the service integration bus appears as a queue manager to the WebSphere MQ network.

The following figure illustrates an example of the message flow from a service integration bus to a WebSphere MQ network over a WebSphere MQ link.

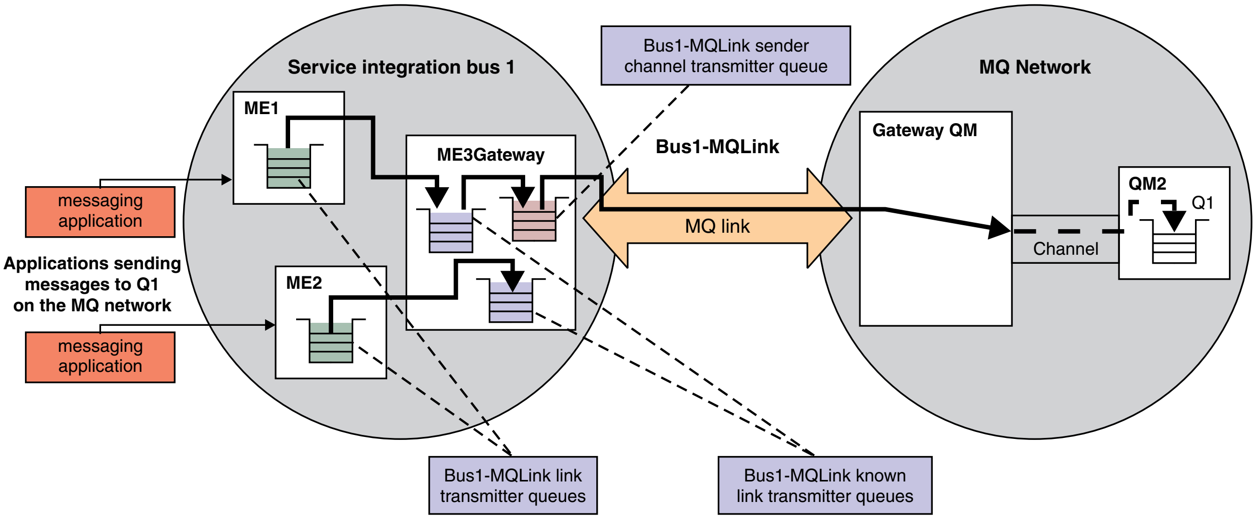

Service integration bus 1
Bus1-MQLink sender channel transmitter queue
MQ Network
ME1
ME3Gateway
Bus1-MQLink
Gateway QM
QM2
Q1
messaging application
MQ link
Channel
Applications sending messages to Q1 on the MQ network
ME2
messaging application
Bus1-MQLink link transmitter queues
Bus1-MQLink known link transmitter queues

Figure 94. Message flow between a service integration bus and a WebSphere MQ network

Point-to-point messaging across multiple buses:

Point-to-point messaging uses queue destinations, where each queue destination represents a message queue.

A service integration queue destination is localized in a particular bus member (application server or server cluster). When a producer sends a message to the queue destination, the service integration bus delivers the message to a messaging engine in that bus member. The messaging engine then delivers the message to a consumer; if necessary, the messaging engine queues the message until a consumer is ready to receive it.

Applications can send messages to a queue destination in a remote bus, as long as a connection between the buses is configured. You can configure a bus to connect to, and exchange messages with, other messaging networks. To do this, you must configure a foreign bus connection. A foreign bus connection encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus. The local bus knows of the destination bus through a foreign bus connection. If the bus that is directly connected to the local bus does not hold the specified destination, more service integration bus links are used to flow the message to the next bus on the route to the destination bus. When the message enters the destination bus, that bus attempts to deliver the message to the intended destination.

To send messages to a destination that is defined in a foreign bus, an application specifies the bus name (that is, the foreign bus) and the destination name in the JMS destination object (queue or topic). You do not need to configure any destination objects in the local bus. Service integration uses the definition of the foreign bus that is configured on the local bus, that is, the foreign bus connection. This definition includes default values for the destination attributes, such as the default quality of service. These default values apply to all destinations in that foreign bus.

You can configure a queue destination as a foreign destination or an alias destination, as described in the information about bus destinations.

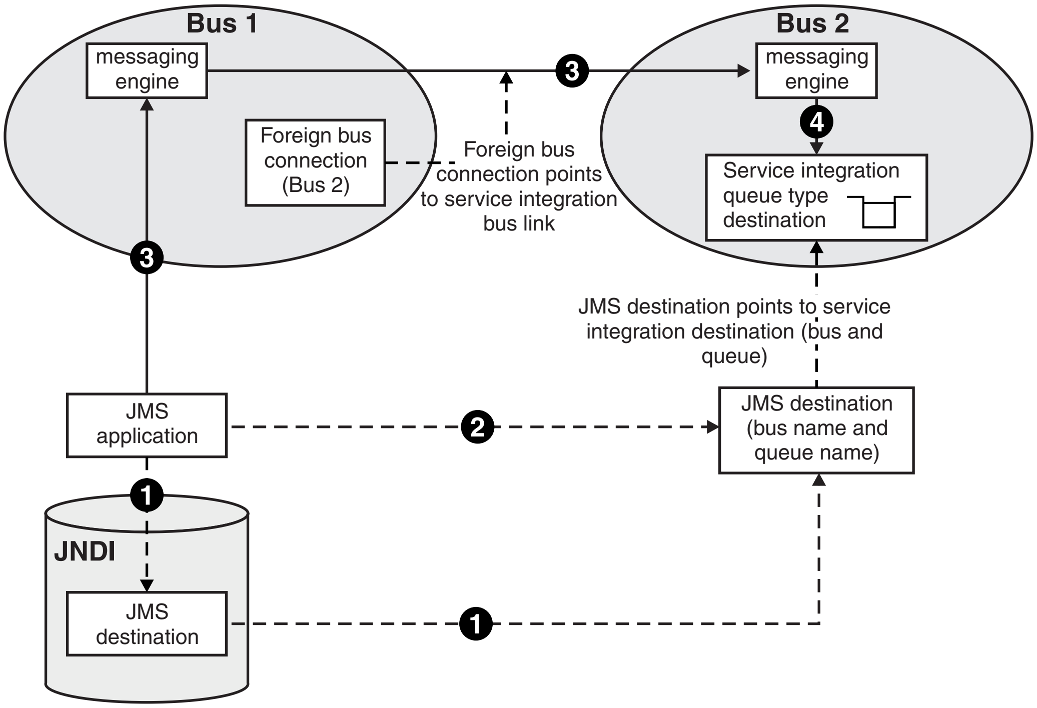
When an application sends messages to a destination and a foreign destination or alias destination is not configured, the destination defaults are derived from the destination defaults that are specified for the foreign bus connection.

For example, an application is connected to Bus 1 (its local bus). The application sends a message to a JMS destination that specifies the bus name Bus 2 and the destination name targetQueue. The message is processed as follows:

- Service integration routes the message to Bus 2 by using the definition in Bus 1 of the foreign bus Bus 2.
- Bus 2 delivers the message to targetQueue by using the definition in Bus 2 of its local queue targetQueue.

In this example, service integration in Bus 1 uses attributes of its definition of the foreign bus, Bus 2, as defaults for its destination targetQueue in the foreign bus. Service integration cannot use configuration information that is scoped to a foreign bus. For example, service integration in Bus 1 is not aware of the Bus 2 definition of targetQueue.

In the following figure, a JMS application connected to Bus 1 creates a producer for a queue in Bus 2. The application uses JNDI to obtain a JMS destination object, which identifies the service integration bus queue in Bus 2. An application can obtain a JMS destination in other ways, for example, from the JMSReplyTo property of a JMS message.


Bus 1
messaging engine
Foreign bus connection (Bus 2)
3
Foreign bus connection points to service integration bus link
3
Bus 2
messaging engine
4
Service integration queue type destination
JMS destination points to service integration destination (bus and queue)
JMS application
2
JMS destination (bus name and queue name)
1
JNDI
JMS destination
1

Key:

1. The JMS application uses JNDI lookup to obtain the JMS destination.
2. The JMS application sends the message to the JMS destination; this is a logical message flow.
3. The local bus, Bus 1, transfers the message from the sending application to the foreign bus, Bus 2, which contains the target destination. Bus 1 applies default properties and destination roles form the foreign bus connection.
4. The foreign bus places the message on the target destination.

Figure 95. Establishing point-to-point messaging between two buses

Knowledge that a destination exists is held only by the bus that hosts that destination. For an application to send messages to a destination in a foreign bus successfully, you must ensure that the destination exists; the local bus cannot verify that the destination exists. If a message arrives through the service integration bus link for a destination that does not exist in the foreign bus, the message is routed using the exception handling configuration of the receiving service integration bus link.

Note: An application cannot consume messages from a destination that is located in a different bus to the one that the application is connected to. Any attempt to create a consumer to a destination on a foreign bus is rejected.

Publish/subscribe messaging across multiple buses:

In service integration, publish/subscribe messaging uses topic space destinations, where each topic space destination represents a set of “publish and subscribe” topics. When multiple buses are connected using a service integration bus link, messages published in a topic space in one bus are accessible to subscribers on a topic space in another bus.

The topic for a specific message (publication) is a property of the message. A service integration topic space destination is not localized in a particular bus member. Service integration maintains a list of subscriptions in the topic space and matches each publication against that list. When a new publication matches one or more subscriptions in the topic space, service integration delivers a copy of the publication to each subscriber; if necessary, service integration can queue the publication message until the subscriber is ready to receive it. When the new publication does not match any subscription, service integration discards the publication.

Subscribers can receive topics that are published in a remote bus, as long as a connection between the buses is configured. You can configure a bus to connect to, and exchange messages with, other messaging networks. To do this, you must configure a foreign bus connection. A foreign bus connection encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus.

To connect topic space destinations in neighboring buses, you configure topic space mappings when you create a foreign bus connection. Each entry in the mapping maps a local topic space destination to a topic space in the foreign bus. Then, any subscribers to topics on the topic space in the local bus can receive messages that are published on those topics in the topic space in the foreign bus. Publish/subscribe applications create producers and subscriptions to topic spaces in their local bus rather than the foreign bus (unlike point-to-point applications and queue destinations). You must configure the topic space mappings so that the messages that the applications publish are routed correctly.

In the following figure, a subscriber in Bus 1 can receive messages that are published in Bus 2. A topic space mapping in Bus 1 enables publications from publishers attached to Topic space 2 in Bus 2 to flow to subscriptions attached to Topic space 1 in Bus 1.

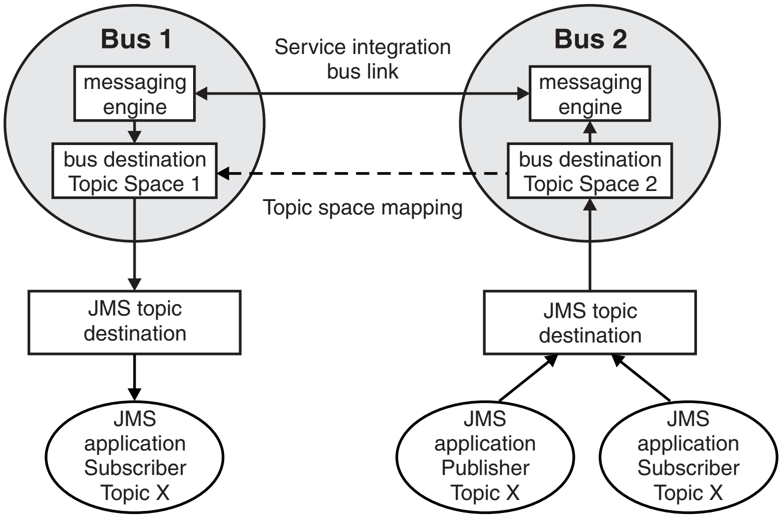

Bus 1
Service integration bus link
Bus 2
messaging engine
messaging engine
bus destination Topic Space 1
bus destination Topic Space 2
Topic space mapping
JMS topic destination
JMS topic destination
JMS application Subscriber Topic X
JMS application Publisher Topic X
JMS application Subscriber Topic X

Figure 96. Publish/subscribe messaging across Bus 1 and Bus 2

If subscriptions exist in both buses and publishers can publish from either bus, a topic space mapping is required in both buses to enable publications to flow between all publishers and subscribers in the two buses.

When connecting topic spaces in more than two buses, there is no restriction on how multiple buses can be connected. However, there is a restriction on how their topic spaces are connected using topic space mappings. For guidance on how to create topic space mappings, see Configuring topic space mappings between service integration buses.

Note: The names of the local and foreign topic spaces do not have to match, but the names of the topics in the local and foreign buses do have to match.

A network of buses can contain loops in its topology, as shown in Figure 97 on page 626. Interconnected topic spaces must follow a hierarchical tree formation. This tree can then be overlaid on the underlying bus topology by using topic space mappings, as shown in Figure 97. It is not permitted for a set of interconnected topic spaces to form a loop across multiple buses, as shown in Figure 98.

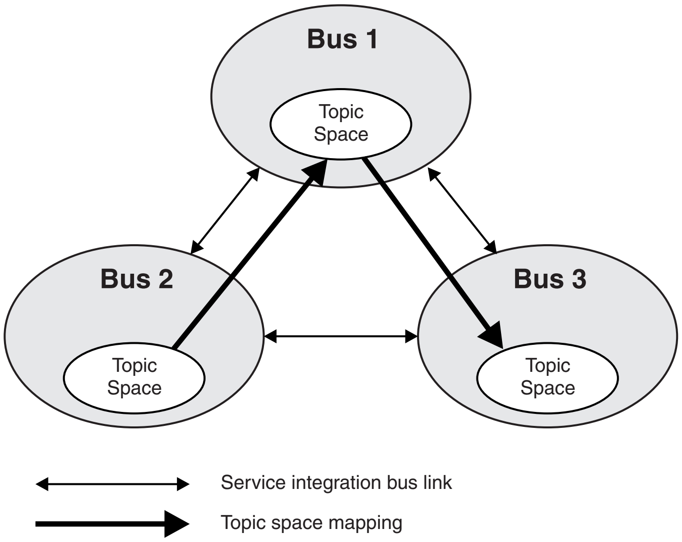


Figure 97. Network of buses with topic spaces connected correctly

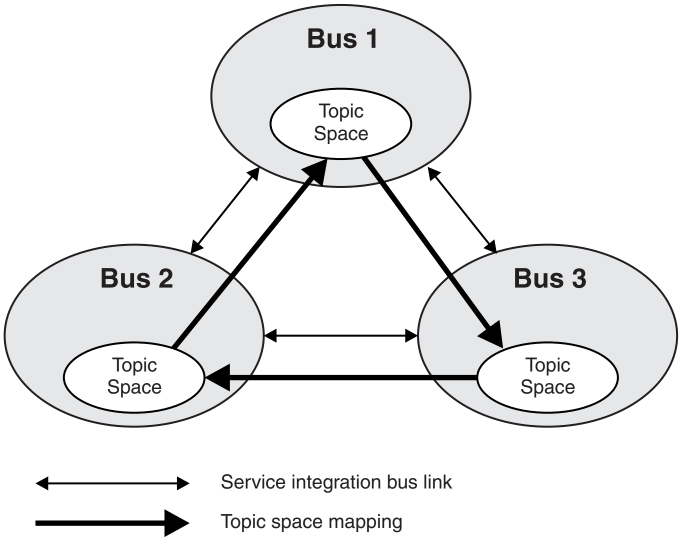


Figure 98. Network of buses with topic spaces connected incorrectly

The correct example in Figure 97 shows that messages published in Bus 2 are automatically flowed through Bus 1 to Bus 3, if Bus 3 has a suitable subscription for those messages. You create a mapping from the topic space in Bus 1 to the topic space in Bus 2, and another mapping from the topic space in Bus 3 to the topic space in Bus 1.

If you also created a mapping from the topic space in Bus 2 to the topic space in Bus 3, as shown in Figure 98, you would create a loop and published messages might enter an indefinite loop and be transmitted continually around the three buses.

If an additional subscription for the messages exists in Bus 2 and a publisher is also connected to Bus 3, topic space mappings are required in the reverse direction to allow messages to flow to all subscriptions in the system, as shown in Figure 99.

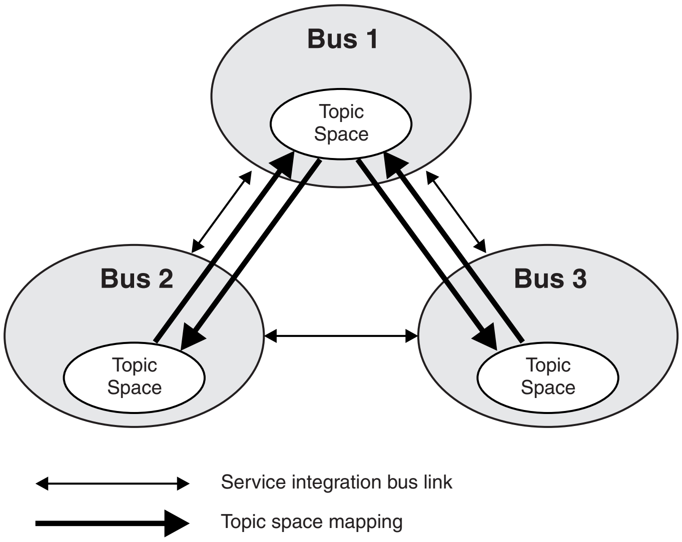


Figure 99. Network of buses with topic space mappings in both directions

Bus topology that links to WebSphere MQ networks:

Service integration buses can link to WebSphere MQ networks. Applications that are connected to a WebSphere MQ queue manager or queue-sharing group can send messages to an application that is attached to a service integration bus, and vice versa.

One way to connect a service integration bus to a WebSphere MQ network is to use a *WebSphere MQ link*. Another way is to use the WebSphere MQ server facility. This topic describes the WebSphere MQ link.

A WebSphere MQ link connects a messaging engine to a WebSphere MQ queue manager or queue-sharing group (known as the *gateway queue manager*) by using sender and receiver channels, thereby connecting the bus and the WebSphere MQ network.

The WebSphere MQ link provides connectivity not just with the messaging engine that hosts the link, but also with the other messaging engines in the bus. All the messaging engines in the bus appear to the WebSphere MQ network as if they were a single queue manager (they inherit the queue manager name from the WebSphere MQ link).

WebSphere MQ links can be used in a number of different configurations as shown in Figure 100 on page 628 and Figure 101 on page 629. A messaging engine can contain multiple WebSphere MQ links.

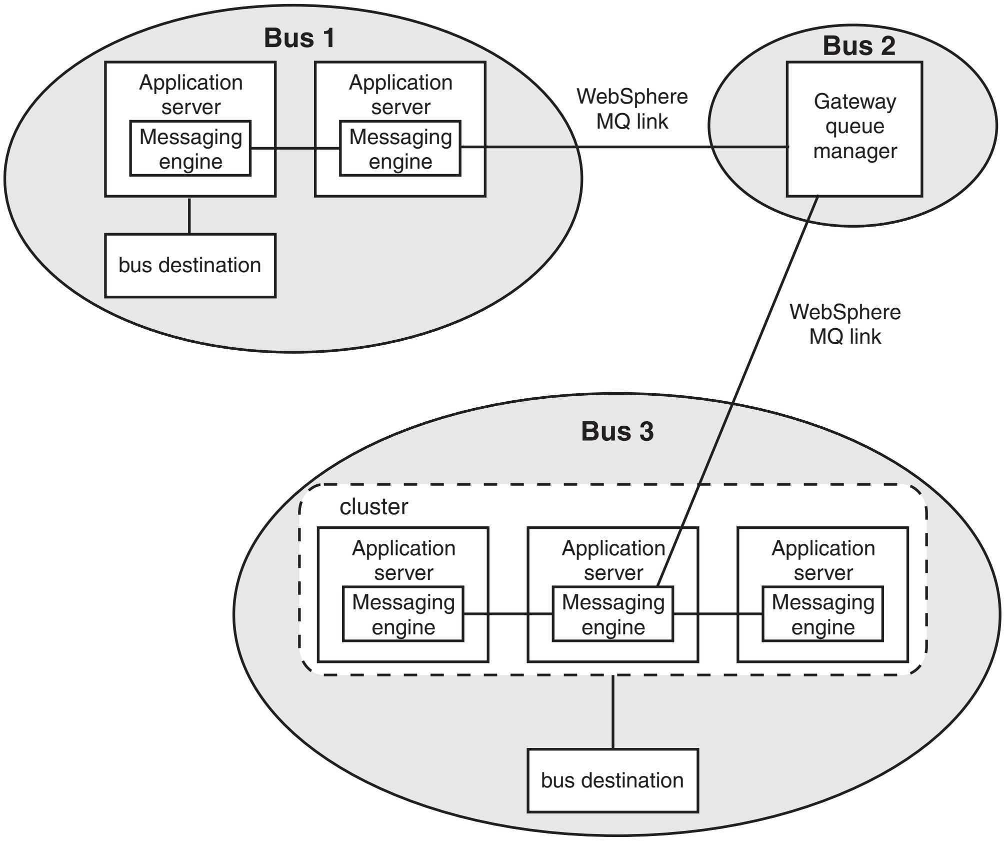


Figure 100. Service integration buses with links to a WebSphere MQ network

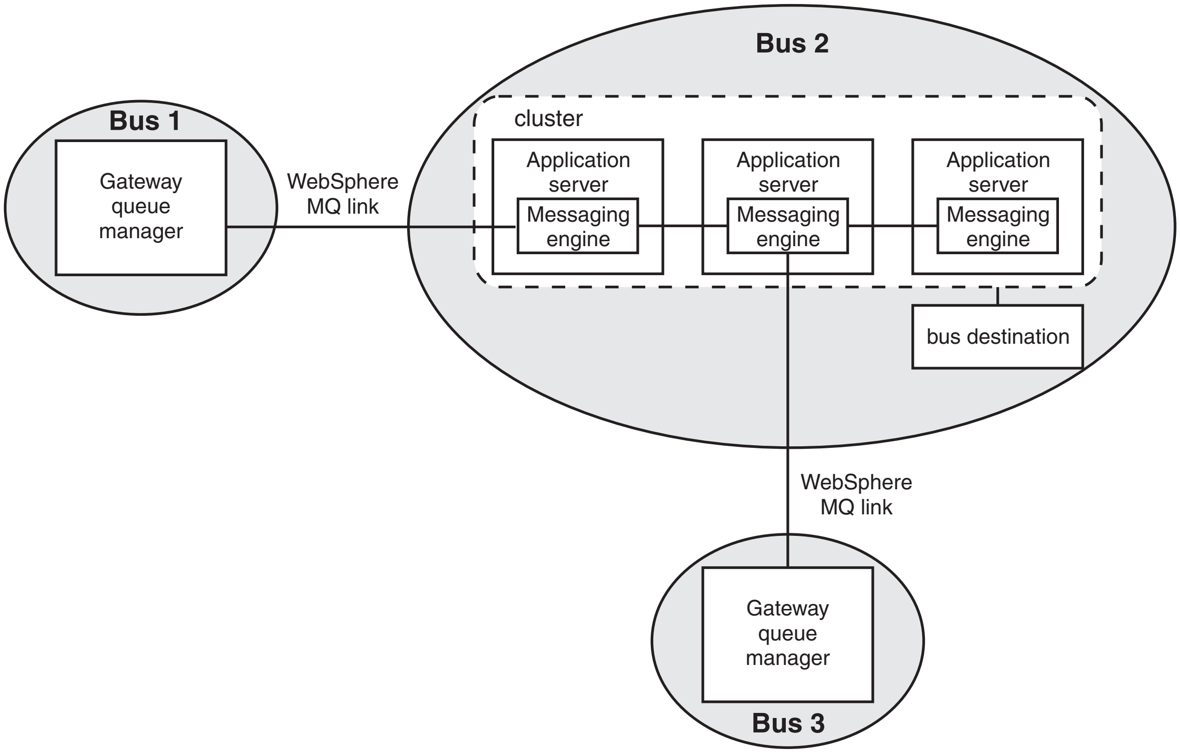


Figure 101. Service integration bus with links to WebSphere MQ networks

You can also use a WebSphere MQ link to form a publish/subscribe bridge that allows publication and subscription between WebSphere Application Server and the WebSphere MQ native publish/subscribe capability.

Direct and indirect routing between service integration buses:

You can use direct or indirect connections to interconnect service integration buses so that all the buses can exchange messages.

Service integration buses can be connected directly or indirectly. For a direct connection, two buses are either connected directly by a single service integration bus link to another service integration bus or by a WebSphere MQ link to a WebSphere MQ queue manager or queue-sharing group (known as the “gateway queue manager”). For an indirect connection, two buses are connected through one or more intermediate buses that are connected in a chain of links.

Each bus must be able to get to every other bus to which it is connected. In this context, the bus you start with is referred to as the local bus, and each bus that it is connected to is referred to as a foreign bus. Information about how messages are routed from the local bus to each foreign bus is stored in the local bus in the routing properties of the foreign bus connection. The routing properties indicate the following information:

- Whether the connection is direct or indirect.
- If the connection is direct, the type of physical link can be either a *service integration bus link* from a messaging engine in the local bus to a messaging engine in the foreign bus, or a *WebSphere MQ link* from a messaging engine in the local bus to the gateway queue manager in the WebSphere MQ network.
- If the connection is indirect, the next foreign bus in the chain that leads to the target bus.

In Figure 1, Bus 1 and Bus 2 are connected directly using a single direct service integration bus link. The messaging engine in Bus 1 is connected to a messaging engine in Bus 2 by using a direct service integration bus link.

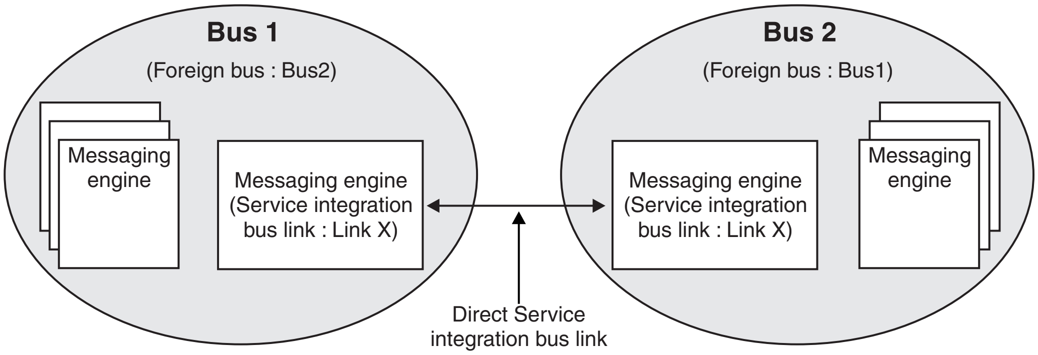

Bus 1
(Foreign bus : Bus2)
Messaging engine
Messaging engine (Service integration bus link : Link X)
Bus 2
(Foreign bus : Bus1)
Messaging engine (Service integration bus link : Link X)
Messaging engine
Direct Service integration bus link

Figure 102. Direct connection between two service integration buses by using a service integration bus link

To connect one bus to another bus through an intermediate bus, or a chain of buses, if the connection between the intermediate bus or the chain of buses and the target bus already exists, you do not require any new physical links. Instead, each foreign bus connection identifies a neighboring bus on the route to the final target bus as the “next hop” in the chain. Each bus in the chain must know about the next hop in the chain to reach the target bus. The local bus uses a foreign bus connection to identify the next bus in the chain to the target bus, and uses its direct physical link to flow messages to that bus. Then, each intermediate bus uses its locally defined foreign bus connection to identify the next bus in the chain until the target bus is reached.

The physical link in the chain can be a service integration bus link or a WebSphere MQ link.

In Figure 2, to route a message from Bus 1 to Bus 3, the message is routed through a link from Bus 1 to Bus 2, then is routed through another link from Bus 2 to Bus 3. Bus 1 has a foreign bus connection that identifies Bus 2 as a direct connection, and a foreign bus connection that identifies Bus 2 as the next foreign bus on the route to Bus 3. Bus 2 has a foreign bus connection that identifies the next hop to the final bus, which in this example, is a direct connection to Bus 3.

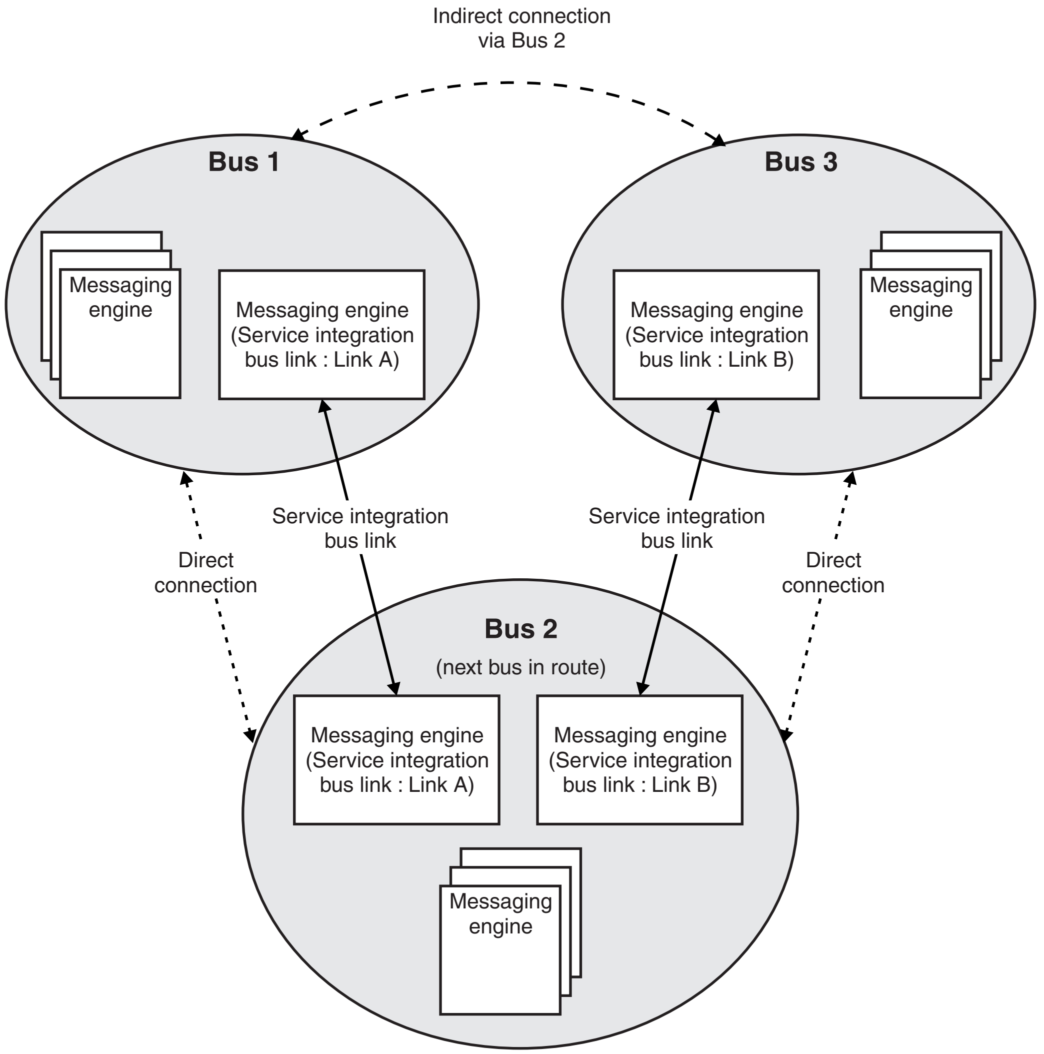


Figure 103. Indirect connection between two service integration buses, by using two direct service integration bus links

The following diagram shows an existing network of three buses, Bus 1, Bus 2 and Bus 3, where a new bus, Bus n, is to be added. Bus n will be connected directly to Bus 1 and indirectly to Buses 2 and 3.

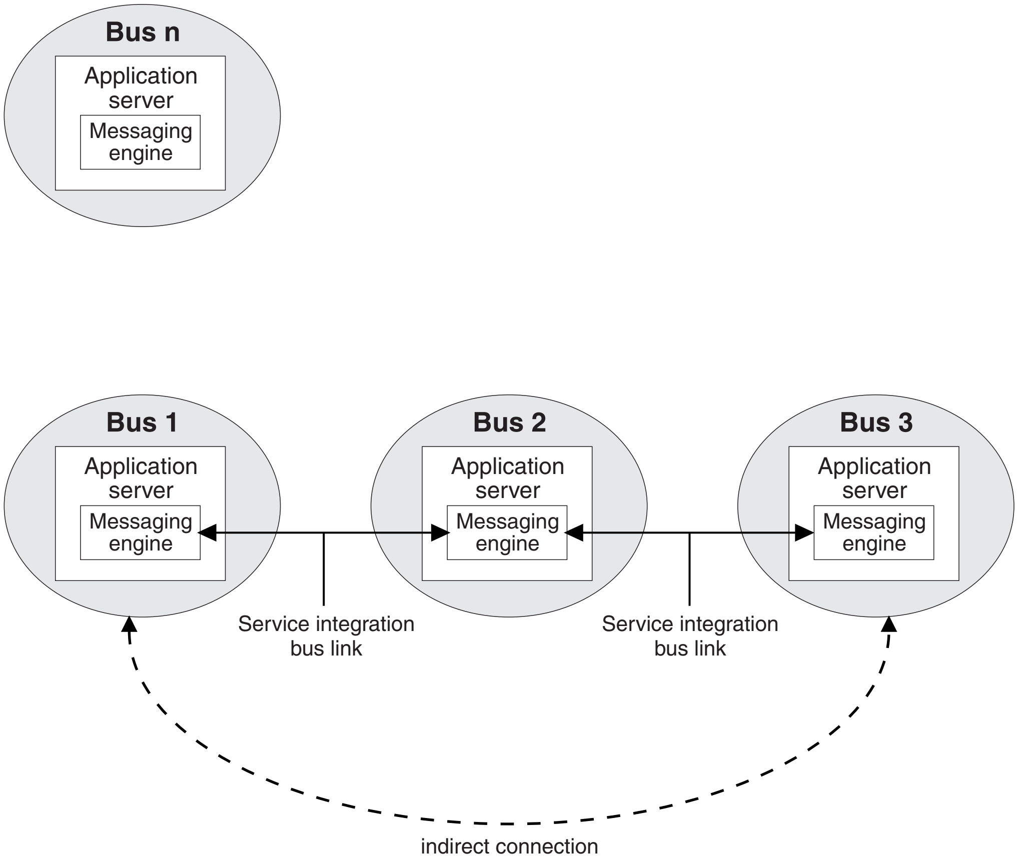


Figure 104. A network before Bus n is added

The following list shows the resources that must be defined to add Bus n to the network, and to allow messages to flow between any of the buses:

- For Bus n, the following resources must be defined:
  - A direct foreign bus connection that represents Bus 1. A service integration bus link between the messaging engine on Bus n and the messaging engine on Bus 1 is created automatically.
  - An indirect foreign bus connection that specifies Bus 1 as the next bus in the chain and Bus 2 as the target bus.
  - An indirect foreign bus connection that specifies Bus 1 as the next bus in the chain and Bus 3 as the target bus.
- For Bus 1, a direct foreign bus connection that represents Bus n. A service integration bus link between the messaging engine on Bus 1 and the messaging engine on Bus n is created automatically. The name of the service integration bus link must exactly match the name of the service integration bus link created in Bus n.
- For Bus 2, an indirect foreign bus connection that specifies Bus 1 as the next bus in the chain and Bus n as the target bus.
- For Bus 3, an indirect foreign bus connection that specifies Bus 2 as the next bus in the chain and Bus n as the target bus.

The following diagram shows the network after Bus n is added. Bus n is connected directly to Bus 1. The messaging engine in Bus n is connected to the messaging engine in Bus 1 by using a direct service integration bus link. There is an indirect connection between Bus n and Bus 2, and between Bus n and Bus 3.

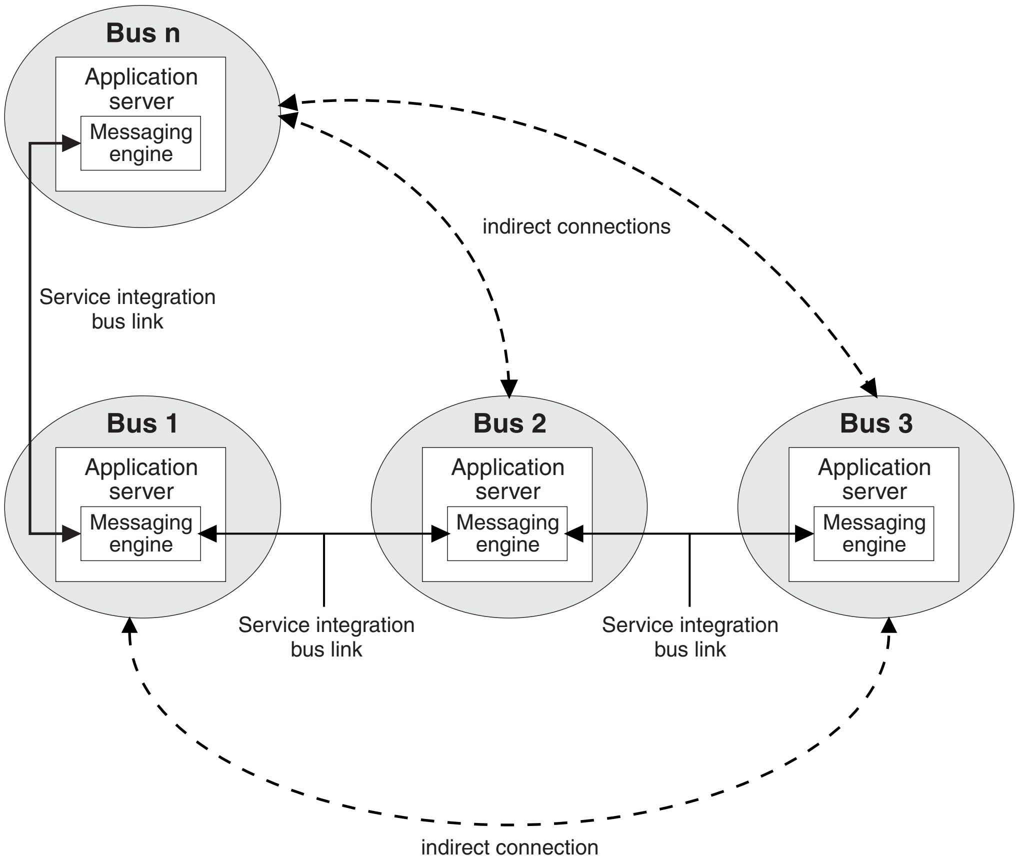


Figure 105. A network after Bus n is added

Bootstrap members

An application inside a client container or application server outside the cell, cannot connect directly a bus inside the cell. This is because the client container or application server outside the cell, does not have access to cell configuration information. Therefore, to connect to a bus, the application must bootstrap to an application server inside the cell. That application server identifies a server inside the cell that is a bus member. Then, the client or application server outside the cell, will be able to connect to the bus inside the cell. A bootstrap member is an application server that is configured to accept requests to bootstrap into the service integration bus. The bootstrap member authenticates a connection request, and directs the request to a suitable bus member. The administrator configures the bootstrap member policy for the bus to determine which types of server can service requests to bootstrap.

The following bootstrap member policies are available:

All members of the cell with the SIB Service enabled
This the default policy. Any server in the cell that has the SIB service enabled can service bootstrap requests.

Bus members and nominated bootstrap members
Only bus members or nominated bootstrap members can service bootstrap requests.

Bus members only
Only bus members can service bootstrap requests.

Unlike a bus member, a bootstrap member does not host a messaging engine and therefore does not provide messaging services. However, an application server can be both a bootstrap member and a bus member.

The administrator can use the administrative console to list bootstrap members. This information is helpful in managing bus topologies, developing applications, and diagnosing problems.

If a server is a member of one bus only, the administrator can choose the bootstrap member policy **Bus members and nominated bootstrap members** to prevent the members of one bus from bootstrapping into another bus.

If a bus topology uses multiple security domains, you can isolate buses and the applications that use them by configuring the bootstrap members so that only a subset of servers can access a bus.

Service integration notification events

You can monitor your service integration environment by using notification events.

Each service integration entity, such as a messaging engine or a WebSphere MQ link, is represented by an *MBean*. MBeans are Java objects that represent Java Management Extensions (JMX) resources. As part of the JMX architecture, the MBeans can have notification events, which you can incorporate into your own system management application to monitor activities, such as the starting up of a messaging engine, across your service integration configuration. Events also enable applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

For more information about the WebSphere Application Server implementation of the JMX framework, see Using wsadmin scripting with Java Management Extensions (JMX).

Each WebSphere Application Server MBean is documented in the Mbean interfaces section of the Information Center. You can navigate to this section by expanding **Reference > Programming interfaces** in the Information Center navigation, and then clicking **Mbean interfaces**, or by doing a search for "Mbean interfaces".

The following service integration MBeans have notification events:
- SIBGatewayLink
- SIBMediationPoint
- SIBMessagingEngine
- SIBMQLink
- SIBMQLinkReceiverChannel
- SIBMQLinkSenderChannel
- SIBPublicationPoint
- SIBQueuePoint
- SIBRemotePublicationPoint
- SIBRemoteQueuePoint

Format of event notifications

Event notifications contain information about the event and its origin. Typically, these notifications are processed by a system management application program tailored to meet the requirements of the enterprise at which it runs. The service integration events have the following attributes:

Table 46. Attributes of service integration events. The first column of the table lists the event attribute names, and the second column provides the description of the attributes.

Attribute name	Attribute description
Message	The message that describes the event.
SequenceNumber	An identifier for the event instance, unique within a run of the application server.
TimeStamp	Timestamp in milliseconds.
Type	Identifies the type of event, for example a messaging engine stop event. The type is defined by constants in the *app_server_root*/web/apidocs/SIBNotificationConstants.html file.
Properties	Specific for each event generated. These are key value pairs of strings that describe aspects of the event, such messaging engine name or number of messages on a destination. The key and value strings are defined by constants in the *app_server_root*/web/apidocs/SIBNotificationConstants.html file. All service integration events will have properties for the bus name and UUID, and the messaging engine name and UUID.

Message reliability levels - JMS delivery mode and service integration quality of service

Messages have a quality of service attribute that you can use to specify how reliable message delivery will be. JMS applications send messages with a JMS delivery mode (`persistent` or `nonpersistent`), then service integration uses JMS connection factory settings to map the JMS delivery mode to a service integration message reliability setting. Additional settings on bus destinations (including foreign destinations and alias destinations) can override this message reliability.

Note: The term message reliability level corresponds to all of the following terms:

- Quality of service (QoS) (service integration messaging)
- Delivery mode (JMS)
- Persistence (WebSphere MQ)

You can specify the following service integration message reliability levels for messages. The options are listed in order of increasing reliability.

`Best effort nonpersistent`
Messages are discarded when a messaging engine stops or fails. Messages might also be discarded if a connection used to send them becomes unavailable or as a result of constrained system resources.

`Express nonpersistent`
Messages are discarded when a messaging engine stops or fails. Messages might also be discarded if a connection used to send them becomes unavailable.

`Reliable nonpersistent`
Messages are discarded when a messaging engine stops or fails.

`Reliable persistent`
Messages might be discarded when a messaging engine fails.

`Assured persistent`
Messages are not discarded.

Note: Higher levels of reliability have higher impacts on performance.

JMS applications send messages with a **`JMS delivery mode`** (`persistent` or `nonpersistent`). Applications specify this delivery mode as a parameter of the JMS send() method, but you can optionally specify a delivery mode that overrides the send() method as an attribute of the JMS destination.

Service integration uses the JMS connection factory settings to map the JMS delivery modes (persistent and nonpersistent) to service integration message reliability levels. You can use this mapping to choose between high performance, high reliability, or something in between. You specify the mapping you require in the JMS connection factory settings. For example, see Default messaging provider unified connection factory [Settings].

Important: If you map the persistent JMS delivery mode to one of the service integration nonpersistent levels (`best effort nonpersistent`, `express nonpersistent` or `reliable nonpersistent`), you risk losing messages in certain circumstances. For example, you risk losing messages at server restart, or when there is heavy workload.

You specify default and maximum service integration reliability levels as attributes of bus destinations (including foreign destinations and alias destinations). You also specify whether the producer-specified reliability overrides the default reliability for the destination: if not, service integration resets the reliability level of the messages to the default reliability for the destination. For alias destinations, you can specify that the reliability setting is inherited from the target destination.

For interoperation with WebSphere MQ, you map the reliability settings for service integration messages to the persistence settings for WebSphere MQ messages. For more information, see Mapping the JMS delivery option and message reliability to and from the WebSphere MQ persistence value.

To help you choose the required reliability level, the following table illustrates the behavior associated with the five levels of reliability.

Note: In addition to setting the selected reliability level, for messages to remain available after the various failures shown in the table with certain reliability, the application must be transactional.

Table 47. Behavior of the five levels of reliability. The five columns in the table lists the five message reliability levels and the corresponding behavior associated with each of the five levels of reliability.

	Best effort nonpersistent	**Express nonpersistent**	**Reliable nonpersistent**	**Reliable persistent**	**Assured persistent**
`JMS delivery mode:`	`Nonpersistent`	`Nonpersistent`	`Nonpersistent`	`Persistent`	`Persistent`
Transactionally atomic:	No, individual messages can be discarded	Yes: messages are not discarded and never retained beyond server restart	Yes: messages are not discarded and never retained beyond server restart	Yes	Yes
Messages hardened:	No	Possibly: when messages build up on a destination	Possibly: when messages build up on a destination	Yes: asynchronously	Yes: synchronously
Messages discarded in normal operation:	Yes	No	No	No	No
Messages duplicated:	No	Possibly: state data can be lost on server failure resulting in duplication	Possibly: state data can be lost on server failure resulting in duplication	Possibly: as deletion from the database is asynchronous with respect to user requests	No
Messages are retained beyond planned shutdown:	No	No	No	Yes: hardened messages are recovered, planned shutdown hardens cached messages	Yes
Messages are retained beyond client comms failure:	No	No	Yes	Yes	Yes

Table 47. Behavior of the five levels of reliability (continued). The five columns in the table lists the five message reliability levels and the corresponding behavior associated with each of the five levels of reliability.

	Best effort nonpersistent	Express nonpersistent	Reliable nonpersistent	Reliable persistent	Assured persistent
Messages are retained beyond engine comms failure:	No	Yes	Yes	Yes	Yes
Messages are retained beyond engine crash:	No	No	No	Possibly: hardened messages are recovered	Yes
Messages are retained beyond backup and restore:	No	No	No	Possibly: hardened messages can be backed up and recovered	Yes

The following provides an explanation of the row headings in the table:

`JMS delivery mode`

For JMS objects such as connection factories and destinations, the mapping between **`JMS delivery mode`** and the reliability settings. The default mapping for the JMS `nonpersistent` delivery mode is `express nonpersistent`. The default mapping for the JMS `persistent` delivery mode is `reliable persistent`.

Transactionally atomic

Whether the message is atomic with respect to other messages produced or consumed within the same transaction. Best effort messages are not transactionally atomic at the time that they are produced with respect to other messages, so if one such message is lost (see the description of `best effort nonpersistent` earlier in this topic for details of how messages might be lost), other messages being processed in the same transaction might still be delivered successfully when the transaction commits (if the transaction is rolled back all operations on messages, irrespective of their reliability, are rolled back). For messages of higher reliability, if a failure occurs that would cause the loss of one of the messages in the transaction, the transaction, and all work being performed under that transaction, will be rolled back, making the operation transactionally atomic.

Messages hardened

Whether messages are written to disk in the data store or the file store. System performance is affected by the frequency with which messages are written to disk, and in general using a file store for your messaging engine can improve performance. Messages of `best effort nonpersistent` reliability are never written to disk, `express nonpersistent` and `reliable nonpersistent` messages are written if messages build up on a destination, whereas `reliable persistent` and `assured persistent` messages are always written to disk.

Messages of `reliable persistent` reliability are written to disk, but this is done asynchronously with respect to the producing application. This allows increased flexibility in scheduling and batching of database updates, which can be used to increase throughput. Messages are not be lost under normal operating conditions, but messages might be lost if a messaging engine fails before this asynchronous write is complete.

Messages of `assured persistent` reliability are written to disk synchronously with respect to the producing application.

If messages are allowed to build up on a destination due to them not being consumed as quickly as they are produced, a messaging engine might choose to write messages to disk in order to manage memory usage.

When a message whose quality of service attribute is better than `best effort nonpersistent` is written to disk, it might also be cached in a memory buffer.

Messages discarded in normal operation

Whether messages are discarded during normal operation.

Note:

If you have a non-transactional message-driven bean, the system either deletes the message when the bean starts, or when the bean completes. If the bean generates an exception, and therefore does not complete, the system takes one of the following actions:

- If the system is configured to delete the message when the bean completes, then the message is despatched to a new instance of the bean, so the message has another opportunity to be processed.
- If the system is configured to delete the message when the bean starts, then the message is lost.

The message is deleted when the bean starts if the quality of service is set to `Best effort nonpersistent`. For all other qualities of service, the message is deleted when the bean completes.

Messages duplicated
Whether messages are duplicated following a server failure.

Messages are retained beyond planned shutdown
Whether messages are retained beyond a planned shutdown or startup.

Messages are retained beyond client comms failure
Whether messages are retained beyond the failure of client-messaging engine communication.

Messages are retained beyond engine comms failure
Whether messages are retained beyond the failure of inter-engine communication.

Messages are retained beyond engine crash
Whether messages are retained beyond the failure of a messaging engine or a server.

Messages are retained beyond backup and restore
Whether messages are retained beyond an online backup and restore process.

Dynamic reloading of configuration files

Updates to configuration information are dynamically passed to the server if you use dynamic reloading of configuration files. These updates are available to a messaging engine even if it is not started.

The information that defines the configuration of service integration buses (including any linked foreign bus connections) and their resources is saved in a set of configuration files. When a server starts up, it reads the current information about service integration from those configuration files. When a messaging engine is started, it uses the information in the server that it is running in.

If the information in the configuration files is changed while the server is running, the server must either be dynamically updated or restarted to use the updated information.

You can enable dynamic reloading of configuration files for servers and for service integration buses, including the configuration information for any linked foreign bus connections. There are different results depending on whether you enable dynamic reloading for either the bus or the server or both:

- If you enable dynamic reloading on the bus, but not on the application server, you will need to restart the server for updates to take effect.
- If you enable dynamic reloading on the server, but not on the bus, then you will need to restart only the messaging engine for the updates to take effect.
- If you enable dynamic reloading on both the bus and the server, then all updates occur automatically without the need for restarts.

To enable dynamic reloading of configuration files on an application server, click **Application servers -> *server_name* -> SIB service** to display the Application Servers window, and select **Configuration reload**

enabled. To ensure that dynamic configuration updates are made to each node, click **System administration -> Console Preferences** to display the Console Preferences window and select **Synchronize changes with nodes**.

To enable dynamic reloading of configuration files for a service integration bus, click **Service integration -> Buses -> *bus_name*** to display the bus details, and select **Configuration reload enabled**.

If you choose not to enable dynamic reloading of configuration files, you must restart the server to pick up changes to the configuration files.

Service integration backup

You should back up your service integration setup on a regular basis so that it can be restored in the event of an unrecoverable failure.

For service integration, these are the main components that you can back up:

Configuration files
The configuration of a service integration setup is stored as XML files. To back up or restore these configuration files, use the relevant command as detailed in Backing up and restoring administrative configuration files. Any backup or restore of a service integration setup must include a backup or restore of the configuration files.

Data stores that are accessed by the messaging engines
Backing up and restoring your data stores is optional. As messages are transient in nature, you might not want to back up or restore the data stores.

- If you do not back up the data stores, and you modified your current configuration since it was last backed up, when you restore the configuration backup be aware that you might lose messages. For example, if you back up the configuration and then create a bus destination, when you restore the configuration backup the bus destination will no longer exist. Any messages for this destination will be deleted when the server that hosted that messaging engine is restarted.
- If you do back up your data stores, you must also back up the configuration files. You must back up or restore the configuration files at the same time as the data store is backed up or restored. Backing up and restoring at the same time maintains the consistency of the system and reduces the possibility of losing or duplicating messages from the time of the backup.

To back up a data store, see Backing up a data store.

File stores that are accessed by the messaging engines
Backing up and restoring your file stores is optional. As messages are transient in nature, you might not want to back up or restore the files. To back up a file store, see Backing up a file store.

If you have multiple servers, you should ideally back up all the servers at the same time, otherwise messages from the time of the backup might be lost or duplicated. You should also minimize message traffic to reduce the possibility of losing or duplicating messages.

When you restart a messaging engine after restoring a backup, you should take steps to minimize loss of messages. See Restoring a data store and recovering its messaging engine .

Chapter 21. Session Initiation Protocol (SIP) applications

This page provides a starting point for finding information about SIP applications, which are Java programs that use at least one Session Initiation Protocol (SIP) servlet written to the JSR 116 specification.

SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging.

SIP in WebSphere Application Server

WebSphere Application Server delivers rich SIP functionality throughout its infrastructure.

Session Initiation Protocol (SIP) has grown considerably since it first became an IETF standard in 1999. SIP was originally intended purely for video and audio but has now grown to become the control protocol for many interactive services, particularly in the peer-to-peer realm. SIP, and the standards surrounding SIP, provide the mechanisms to look up, negotiate, and manage connections to peers on any network over any other protocol.

This product includes support for SIP Servlet Specification 1.1, also referred to as Java Specification Request (JSR) 289. The SIP Servlet Specification provides the Java API standards for Session Initiation Protocol (SIP). JSR 289 is an update to the existing SIP Servlet specification that addresses new requirements determined by industry users. This product continues to support the SIP Servlet 1.0 specification.

WebSphere Application Server also provides tooling for the development environment and high performing Edge Components to handle distributed application environments.

In the application server, the web container and SIP container are converged and are able to share session management, security and other attributes. In this model, an application that includes SIP servlets, HTTP servlets, and portlets can seamlessly interact, regardless of the protocol.

It's important to note that the SIP function in the proxy server is stateless. The SIP RFC defines two types of proxy servers, one stateful and one stateless. Normally, a SIP proxy is a stateful instance and stateless proxies are specified as such. A stateful proxy participates in the call flows and is implemented using SIP servlets.

The stateless SIP proxy functionality in the proxy server allows the proxy to handle the workload, routing, and session affinity needs of the SIP container with less complexity. Being stateless, the proxy server can be fronted by a simple IP sprayer. If a proxy server fails, the affinity is to the container and not to the proxy itself so there is one less potential failure along the message flow.

SIP Infrastructure

The SIP infrastructure is a multi-tiered architecture made up of SIP containers, SIP proxies and an IP sprayer. The SIP container is a general purpose SIP application server. The SIP infrastructure consists of:

- SIP container – web container extension that implements JSR 289.
- SIP proxy – Stateless edge device that handles I/O concentration, load balancing, and other functions, in a similar manner to the reverse HTTP proxy. This is not the same as the SIP proxy defined by RFC 3261.

SIP is a key element for many new applications, especially when converged with HTTP, including:

- Click-To-Call
- Voice over IP
- Third Party Call Control and Call Monitoring

- Presence and Instance Messaging

SIP applications

A *SIP application* is a Java program that uses at least one Session Initiation Protocol (SIP) servlet.

A SIP servlet is a Java-based application component that is managed by a SIP servlet container and that performs SIP signaling. Like other Java-based components, servlets are platform-independent Java classes that are compiled to platform-neutral bytecode that can be loaded dynamically into and run by a Java-enabled SIP application server. Containers, sometimes called servlet engines, are server extensions that handle servlet interactions. SIP servlets interact with clients by exchanging request and response messages through the servlet container.

SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging. “Presence” in this context refers to user status such as “Active,” “Away,” or “Do not disturb.” The standard that defines a programming model for writing SIP-based servlet applications is JSR 116.

SIP container

This product complies with the following SIP standards:

IETF

JCP

For a complete list of the supported Internet Engineering Task Force (IETF) and Java Community Process (JCP) industry standards, see the “Compliance with industry SIP standards” topic linked later in this section.

SIP industry standards compliance

The product implementation of Session Initiation Protocol (SIP) complies with industry standards for both a SIP container and SIP applications.

SIP container

This product complies with the following SIP standards:

IETF

JCP

This product also complies with the Internet Engineering Task Force (IETF) and Java Community Process (JCP) industry standards for SIP. The following table contains a list of the IETF and JCP standards.

Table 48. WebSphere Application Server complies with these SIP standards.

This product complies with SIP standards as noted in the following table.

Standard	Description
JR116	SIP: SIP Servlet API
JR289	SIP: SIP Servlet API v1.1
RFC 2543	SIP: Session Initiation Protocol
RFC 3261	SIP: Session Initiation Protocol
RFC 3262	Reliability of provisional responses in SIP
RFC 3263	Locating SIP servers
RFC 3265	SIP-specific event notification
RFC 3311	The SIP UPDATE Method

Table 48. WebSphere Application Server complies with these SIP standards (continued).

This product complies with SIP standards as noted in the following table.

Standard	Description
RFC 3325	Private Extensions to the SIP for asserted identity within trusted networks
RFC 3326	The Reason Header field for the SIP
RFC 3515	The SIP Refer method
RFC 3581	The SIP Extension for Symmetric Response Routing
RFC 3824	Using E.164 numbers with the SIP
RFC 3891	The SIP Replace header
RFC 3903	SIP Extension for event state publication
RFC 3911	The SIP Join header
RFC 4475	SIP torture test messages
RFC 5057	Multiple dialog usages in SIP
RFC 5626	Managing Client-Initiated Connections in SIP **Note:** The SIP server can act as a proxy or registrar as specified in sections, 5, 6, and 7 of RFC 5626. RFC 5626 is an extension to RFC 3261. The SIP server has full support for RFC 3261. However, support for RFC 5626 comes with the following limitations: • The SIP server can act as a User Agent, as defined in RFC 3261; however, it cannot act as a User Agent as defined in section 4 of RFC 5626. • The SIP server does not support STUN keepalives, as specified in RFC 5626.
RFC 5658	Addressing Record-Route issues in SIP

SIP applications

This product complies with standards for SIP applications.

Table 49. Compliance with standards for SIP applications.

This product complies with standards for SIP applications as noted in the following table.

Standard	Description
RFC 2848	The PINT Service Protocol: Extensions to SIP and Session Description Protocol (SDP) for internet protocol (IP) access to telephone call services
RFC 2976	The SIP INFO method
RFC 3050	Common gateway interface for SIP
RFC 3087	Control of service context using SIP request-URI
RFC 3264	An offer and answer model with SDP
RFC 3266	Support for IPv6 in SDP
RFC 3312	Integration of resource management and SIP
RFC 3313	Private SIP extensions for media authorization
RFC 3319	Dynamic Host Configuration Protocol (DHCPv6) options for SIP servers
RFC 3327	SIP Extension Header field for registering non-adjacent contacts
RFC 3372	SIP for telephones (SIP-T): context and architectures
RFC 3398	Integrated Services Digital Network (ISDN) User Part (ISUP) to SIP mapping
RFC 3428	SIP extension for instant messaging

Table 49. Compliance with standards for SIP applications (continued).

This product complies with standards for SIP applications as noted in the following table.

Standard	Description
RFC 3455	Private Header (P-Header) extensions to the SIP for the 3rd-Generation Partnership Project (3GPP)
RFC 3578	Mapping of Integrated Services Digital Network (ISDN) User Part (ISUP) overlap signaling to the SIP
RFC 3603	Private SIP proxy-to-proxy extensions for supporting the PacketCable distributed call signaling architecture
RFC 3608	SIP Extension Header field for service route discovery during registration
RFC 3665	SIP basic call flow examples
RFC 3666	SIP Public Switched Telephone Network (PSTN) call flows
RFC 3680	A SIP event package for registrations
RFC 3725	Best current practices for third-party call control (3pcc) in the SIP
RFC 3840	Indicating user agent capabilities in the SIP
RFC 3842	A message summary and message waiting indication event package for the SIP
RFC 3856	A presence event package for the SIP
RFC 3857	A watcher information event template package for the SIP
RFC 3959	The early session disposition type for the SIP
RFC 3960	Early media and ringing tone generation in the SIP
RFC 3976	Interworking SIP and intelligent network (IN) applications
RFC 4032	Update to the SIP preconditions framework
RFC 4092	Usage of the SDP Alternative Network Address Types (ANAT) semantics in the SIP
RFC 4117	Transcoding services invocation in the SIP using third-party call control (3pcc)
RFC 4235	An invite-initiated dialog event package for the SIP
RFC 4240	Basic network media services with SIP
RFC 4353	A framework for conferencing with the SIP
RFC 4354	A SIP event package and data format for various settings in support for the push-to-talk over cellular (PoC) service
RFC 4411	Extending the SIP Reason Header for preemption events
RFC 4457	The SIP P-user-database Private-Header (P-Header)
RFC 4458	SIP URIs for applications such as voicemail and interactive voice response (IVR)
RFC 4483	A mechanism for content indirection in SIP messages
RFC 4497	Interworking between the SIP and QSIG
RFC 4508	Conveying feature tags with the SIP REFER method

Runtime considerations for SIP application developers

You should consider certain product runtime behaviors when you are writing Session Initiation Protocol (SIP) applications.

Container may accept non-SIP URI schemes

The SIP container will not reject a message if it doesn't recognize the scheme in the request Uniform Resource Indicator (URI) because the container cannot know which URI schemes are supported by the applications. SIP elements may support a request URI with a scheme other than `sip` or `sips`, for example, the pres: scheme has a particular meaning for presence servers, but the container does not recognize it. It is up to the application to determine whether to accept or to reject a specific scheme. SIP elements may translate non-SIP URIs using any mechanism available, resulting in SIP URIs, SIPS URIs, or other schemes, like the tel URI scheme of RFC 2806 [9].

trns: When a SIP application sends a request to a SIP URI over Transport Layer Security (TLS) in version 6.1, the request URI scheme changes from "sip" to "sips." In the current version, the scheme does not change. You can reverse the new behavior by changing the application code. With a "sips" URI, the behavior remains the same after upgrading from version 6.1 to 7.0 or above. See the information center topic Premigration considerations for more information.

Invoking session listener events

SipSessionListener and SipApplicationSessionListener events are invoked only if an application requests the corresponding session object. You do this by using in your application the method shown in Table 50.

Table 50. Methods that invoke session listener events.

This table lists the methods that invoke session listener events.

Event	Method
SipSessionListener	getSession()
SipApplicationSessionListener	getApplicationSession()

Session activation and passivation

During normal operation, this product never migrates a session from one server to another. Session migration occurs only as a result of a server failure. Therefore the SipSessionActivationListener method's passivation callback is never invoked. However, the activation callback is invoked when a failure forces session failover to a different server.

External resources

If a SIP application performs intensive I/O or accesses an external database, it may be blocked for several milliseconds. If possible, use asynchronous APIs for these resources . Under stress, a blocked SIP application may trigger a Request Timeout or re-transmission.

SIP application attributes

Avoid hanging large objects or BLOBs as SIP Session attributes (via SIPSession.setAttribute API). This may damage the overall performance when combined with high availability (HA). The same recommendation applies for SIPApplicationSession.setAttribute. In most cases, the large object can be replaced by several simple or composed strings.

SIP IBM Rational Application Developer for WebSphere framework

This page provides information about the SIP IBM Rational Application Developer for WebSphere framework.

WebSphere Application Server includes IBM Rational Application Developer for WebSphere to meet all the basic development needs for Java EE applications. Included in IBM Rational Application Developer for WebSphere is support for developing SIP servlet applications. IBM Rational Application Developer for WebSphere provides graphical deployment descriptor editors and basic wizards to get you started writing SIP servlets.

IBM Rational Application Developer for WebSphere also includes many other pieces that integrate well in WebSphere Application Server deployments, such as the Unit Test Environment, which provides the

WebSphere Application Server servlet container to run SIP servlets in the development phase of the product, as well as tools for server automation and application packaging.

IBM Rational Application Developer for WebSphere supports:
- SIP servlet development (JSR 116)
- Converged SIP/HTTP applications
- Import/Export SAR packages
- SIP samples (call forward, call block, third party call)

SIP servlets

This topic describes SIP servlets.

The SIP Servlet 1.0 specification (JSR 116) is standardized through Java Specification Request (JSR) 116. The idea behind the specification is to provide a Java application programming interface (API) similar to HTTP servlets, which provides an easy-to-use SIP programming model. Like the popular HTTP servlet programming model, some flexibility is limited to optimize ease-of-use and time-to-value.

However, the SIP Servlet API is different in many ways from HTTP servlets because the protocol is so different. While SIP is a request-response protocol, there is not necessarily only one response to every one request. This complexity and a need for a high performing solution meant that it was easier to make the SIP servlets natively asynchronous. Also, unlike HTTP servlets, the programming model for SIP servlets sought to make client requests easy to create alongside the other logic being written because many applications act as a client or proxy to other servers or proxies.

SipServlet requests

Like HTTP servlets, each SIP servlet extends a base javax.servlet.sip.SipServlet class. All messages come in through the service method, which you can extend. However, because there is not a one-to-one mapping of requests to responses in SIP, the suggested practice is to extend the doRequest or doResponse methods instead. When extending the doRequest or doResponse methods, it is important to call the extended method for the processing to complete.

Each request method, which the specification must support, has a doxxx method just like HTTP. In HTTP, methods such as doGet and doPost exist for GET and POST requests. In SIP, doInvite, doAck, doOptions, doBye, doCancel, doRegister, doSubscribe, doNotify, doMessage, doInfo, and doPrack methods exist for each SIP request method.

Unlike an HTTP servlet, SIP servlets have methods for each of the response types that are supported. So, SIP servlets include the doProvisionalResponse, doSuccessResponse, doRedirectResponse, and doErrorResponse responses. Specifically, the provisional responses (1xx responses) are used to indicate status, the success responses (2xx responses) are used to indicate a successful completion of the transaction, the redirect responses (3xx responses) are used to redirect the client to a moved resource or entity, and the error responses (4xx, 5xx, and 6xx responses) are used to indicate a failure or a specific error condition. These types of response messages are similar to HTTP, but because the SIP Servlet programming model includes a client programming model, it is necessary to have responses handled programmatically as well.

Clarifications of JSR 116

JSR 289 has made some clarifications to JSR 116, as follows:
- JSR 289 Section 4.1.3: Contact Header Field
- JSR 289 Section 5.2: Implicit Transaction State
- JSR 289 Section 5.8: Accessibility of SIP Servlet Messages

SIP SipServletRequest and SipServletResponse classes:

The SipServletRequest and SipServletResponse classes are similar to the HttpServletRequest and HttpServletResponse classes.

SipServletRequest and SipServletResponse classes

Each class gives you the capability to access the headers in the SIP message and manipulate them. Because of the asynchronous nature of the requests and responses, this class is also the place to create new responses for the requests. When you extend the doInvite method, only the SipServletRequest class is passed to the method. To send a response to the client, you must call the createResponse method on the Request object to create a response. For example:

```
protected void doInvite(SipServletRequest req) throws
 javax.servlet.ServletException, java.io.IOException {

 //send back a provisional Trying response
 SipServletResponse resp = req.createResponse(100);
resp.send();
```

Because of their asynchronous nature, SIP servlets can seem complicated. However, something as simple as the previous code sample sends a response to a client.

Here is a more complex example of a SIP servlet. With the following method included in a SIP servlet, the servlet blocks all of the calls that do not come from the example.com domain.

```
protected void doInvite(SipServletRequest req) throws
 javax.servlet.ServletException, java.io.IOException {

//check to make sure that the URI is a SIP URI
      if (req.getFrom().getURI().isSipURI()){
                 SipURI uri = (SipURI)req.getFrom.getURI();
                 if (!uri.getHost().equals("example.com")) {
                    //send forbidden response for calls outside domain
                    req.createResponse(SipServletResponse.SC_FORBIDDEN).send();
                    return;
                 }
              }
              //proxy all other requests on to their original destination
              req.getProxy().proxyTo(req.getRequestURI);
        }
```

SIP SipSession and SipApplicationSession classes:

Possibly the most complex portions of the SIP Servlet 1.0 specification are the SipSession and SipApplicationSession classes.

SIP SipSession and SipApplicationSession classes

Both of these classes have some useful purposes and can act as the primary place to store data in applications that are designed for distributed or highly available environments.

The SipSession class is the best representative of a specific point-to-point communication between two entities and is the closest to the HttpSession object. Because historically no proxying or forking existed for the HTTP request in HTTP servlets, the need for something higher than a single point-to-point session did not exist. However, even HTTP users can see the growing need for this type of function since portlets began essentially forking HTTP requests. The SIP users expect the proxying and forking activities that require multiple layers of SIP session management. The SipSession class is the lowest point-to-point layer.

The SipApplicationSession class represents the higher layer of SIP session management. One SipApplicationSession class can own one or more SipSession objects. However, each SipSession class can be related to one SipSession object only. The SipApplicationSession class also supports the attachment of any number of other protocol sessions. Currently, only HTTP sessions are supported by any implementations. The SipApplicationSession class has a getSessions method, which takes the requested protocol type as an argument.

You might find it useful for many applications to combine HTTP and SIP. For example, you might use this approach to tie together HTTP and SIP sessions to monitor a phone call or to start a phone call through a rich HTTP graphical user interface.

Example: SIP servlet simple proxy:

This is a servlet example of a simple proxy.

Simple proxy

```
import java.io.IOException;

import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.sip.Proxy;
import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipSession;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.URI;


public class SimpleProxy extends SipServlet implements Servlet {


 final static  private String  SHUTDOWN_KEY = new String("shutdown");
 final static  private String  STATE_KEY = new String("state");
 final static  private int   INVITE_RECEIVED = 1;

 /* (non-Java-doc)
  * @see javax.servlet.sip.SipServlet#SipServlet()
  */
 public SimpleProxy() {
  super();
 }

 /* (non-Javadoc)
  * @see javax.servlet.sip.SipServlet#doInvite(javax.servlet.sip.SipServletRequest)
  */
 protected void doInvite(SipServletRequest request) throws ServletException,
   IOException {

  //log("SimpleProxy: doInvite: TOP");

  try {
   if (request.isInitial() == true)
   {
    // This should cause the sip session to be created. This sample only uses the session on receiving
    // a BYE but the Tivoli performance viewer can be used to track the creation of calls by viewing the
    // active session count.
    Integer state = new Integer(INVITE_RECEIVED);
    SipSession session = request.getSession();
    session.setAttribute(STATE_KEY, state);
             log("SimpleProxy: doInvite: setting attribute");

    Proxy proxy = request.getProxy();

    SipFactory sipFactory = (SipFactory) getServletContext().getAttribute(SIP_FACTORY);
```

```
        if (sipFactory == null) {
            throw new ServletException("No SipFactory in context");
        }

        String callingNumber = request.getTo().toString();
        if (callingNumber != null)
        {
            String destStr = format_lookup(callingNumber);
            URI dest = sipFactory.createURI(destStr);

            //log("SimpleProxy: doInvite: Proxying to dest URI ="  + dest.toString());

         if (((SipURI)request.getRequestURI()).getTransportParam() != null)
          ((SipURI)dest).setTransportParam(((SipURI)request.getRequestURI()).getTransportParam());

         proxy.setRecordRoute(true);
   proxy.proxyTo(dest);
        }
   else
   {
            //log("SimpleProxy: doInvite: Request is invalid. Did not contain a To: field.");
    SipServletResponse sipresponse = request.createResponse(400);
    sipresponse.send();
   }
  }
  else
  {
        //log("SimpleProxy: doInvite: target refresh, let container handle invite");
   super.doInvite(request);
  }
 }
 catch (Exception e){
  e.printStackTrace();
 }
}

/* (non-Javadoc)
 * @see javax.servlet.sip.SipServlet#doResponse(javax.servlet.sip.SipServletResponse)
 */
protected void doResponse(SipServletResponse response) throws ServletException,
  IOException {
  super.doResponse(response);

  // Example of using the session object to store session state.
 SipSession session = response.getSession();
 if (session.getAttribute(SHUTDOWN_KEY) != null)
 {
  //log("SimpleProxy: doResponse: invalidating session");
  session.invalidate();
 }
}

/* (non-Javadoc)
 * @see javax.servlet.sip.SipServlet#doBye(javax.servlet.sip.SipServletRequest)
 */
protected void doBye(SipServletRequest request) throws ServletException,
  IOException {

 SipSession session = request.getSession();
 session.setAttribute(SHUTDOWN_KEY, new Boolean(true));

      //log("SimpleProxy: doBye: invalidate session when responses is received.");
 super.doBye(request);
}

protected String format_lookup(String toFormat){
 int start_index = toFormat.indexOf('<') + 1;
```

```
  int end_index = toFormat.indexOf('>');

  if(start_index == 0){
   //don't worry about it
  }
  if(end_index == -1){
   end_index = toFormat.length();
  }

  return toFormat.substring(start_index, end_index);
 }
}
```

Example: SIP servlet SendOnServlet class:

The SendOnServlet class is a simple SIP servlet that would perform the basic function of being called on each INVITE and sending the request on from there.

SendOnServlet class

Function could easily be inserted to log this invite request or reject the INVITE based on some specific criteria.

```
package com.example;
import java.io.IOException;
import javax.servlet.sip.*;
import java.servlet.ServletException;
public class SendOnServlet extends SipServlet {
  public void doInvite(SipServletRequest req)
       throws ServletException, java.io.IOException {
     //send on the request
     req.getProxy().proxyTo(req.getRequestURI);
  }
}
```

The doInvite method could be altered to do something such as reject the invite for some specific criteria simply. In the following example, all requests from domains outside of example.com will be rejected with a Forbidden response.

```
   public void doInvite(SipServletRequest req)
throws ServletException, java.io.IOException {
if (req.getFrom().getURI().isSipURI()){
    SipURI uri = (SipURI)req.getFrom.getURI();
    if (!uri.getHost().equals("example.com")) {
  //send forbidden response for calls outside domain
req.createResponse(SipServletResponse.SC_FORBIDDEN, "Calls outside example.com not accepted").send();
return;
    }
 }
 //proxy all other requests on to their original destination
 req.getProxy().proxyTo(req.getRequestURI());
   }


SendOnServlet deployment descriptor:
<sip-app>
    <display-name>Send-on Servlet</display-name>
    <servlet>
        <servlet-name>SendOnServlet</servlet-name>
        <servlet-class>com.example.SendOnServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>SendOnServlet</servlet-name>
        <pattern>
              <equal>
```

```
                <var>request.method</var>
                <value>INVITE</value>
              </equal>
        </pattern>
    </servlet-mapping>
</sip-app>
```

Example: SIP servlet Proxy servlet class:

Proxy servlet class

After the initial INVITE, this application will be called on every subsequent SIP message. For each Request and Response, this class will simply print out the action and who it is to or from.

```
package com.example;
import java.io.IOException;
import javax.servlet.sip.*;
import java.servlet.ServletException;
public class ProxyServlet extends SipServlet {
  public void doInvite(SipServletRequest req)
       throws ServletException, java.io.IOException {
     //get the Proxy
     Proxy p=req.getProxy();
     //turn on supervised mode so that all events come through us
     //The default on this is true but it is set to emphasize the function.
     p.setSupervised(true);
     //set record route so we see the ACK, BYE, and OK
     p.setRecordRoute(true);
     //proxy on the request
     p.proxyTo(req.getRequestURI());
  }
public void doRequest(SipServletRequest req)
     throws ServletException, java.io.IOException {
   System.out.println(req.getMethod()+" Request from "+req.getFrom().getDisplayName());
   super.doRequest(req);
}
public void doResponse(SipServletResponse resp)
     throws ServletException, java.io.IOException {
   System.out.println(resp.getReasonPhrase()+" Response from "+resp.getTo().getDisplayName());
   super.doResponse(resp);
}
}

Proxy deployment descriptor
<sip-app>
    <display-name>ProxyServlet</display-name>
    <servlet>
        <servlet-name>ProxyServlet</servlet-name>
        <servlet-class>com.example.ProxyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ProxyServlet</servlet-name>
        <pattern>
              <equal>
                <var>request.method</var>
                <value>INVITE</value>
              </equal>
        </pattern>
    </servlet-mapping>
</sip-app>
```

JSR 289 overview:

Version 8.5 includes support for SIP Servlet Specification 1.1, also referred to as Java Specification Request (JSR) 289.

The SIP Servlet Specification provides the Java API standards for Session Initiation Protocol (SIP). JSR 289 is an update to the existing SIP Servlet specification that addresses new requirements determined by industry users.

SIP is a signaling protocol used for creating, modifying, and terminating IP communication sessions such as telephony and presence applications. SIP is not limited to voice communication and can mediate any type of communication session, such as multimedia.

The following is a brief description of new features available in the JSR 289 specification.

- Application router for application selection

  Application routing enables developers to build complex services out of smaller applications. On initial requests the container calls the application router to determine which application to invoke based on the type of request. The application router is the central hub for selecting application order. See the topic on configuring a SIP application router for more information.
- Annotation-based programming

  Annotations provide a fast way to develop applications by embedding metadata directly in applications. For example, you can use the `@SipServlet` annotation to indicate that a class is a SIP servlet. The `@SipApplication` is a package level annotation. All servlets in the package belong to the same application unless the servlet uses `@SipServlet(applicationName)`. For more information on annotations, see section 18 of the JSR 289.
- Converged applications

  JSR 289 provides a new, standardized mechanism for building converged applications. A converged application contains SIP servlet components and other Java EE components, like HTTP servlets and enterprise beans. The specification includes two new classes to support convergence.
  - `ConvergedHttpSession` is an extension to HttpSession for converged applications.
  - `SipSessionUtil` handles session management for converged applications.

  For more information on converged applications, see section 13 of the JSR 289.
- Back-to-back user agent (B2BUA) APIs

  JSR 289 simplifies the B2BUA pattern in applications with the use of the B2BUA helper class. The B2BUA is a frequently used application pattern. The B2BUA acts as an endpoint for two or more dialogs and forwards requests and responses between those dialogs. The B2BUA helper has the ability to create a copy of an incoming request. It also automatically maintains links between sessions on both sides of the B2BUA. For more information on B2BUAs, see section 12 of the JSR 289.

Note: Support for the Session Initiation Protocol (SIP) session key-based targeting mechanism that is described in JSR 289 section 15.11.2 is only supported in stand-alone environments. The session key-based targeting mechanism is not supported in clustered environments. Alternatively, you can use one of the other targeting mechanisms that are described in JSR 289 within a clustered environment:

- Encode the URI mechanism that is described in section 15.11.3.
- Join and replace the targeting mechanism that is described in section 15.11.4.

SIP application router:

The SIP application router is used by the SIP container to select the order in which applications are run within the container.

The SIP container can invoke multiple applications in order to deploy a complete service or function. This modular and compositional approach makes it easier for application developers to develop new applications. The modular applications can be more easily combined and managed, while individual application implementations remain independent.

The application router is responsible for selecting the correct applications in the correct order to service an incoming message. An application router is required for a container to function, but it is a separate logical entity from the container. The application router is based on the JSR 289 specification. See the specification for more details about the application router function.

The default application router (DAR) can be configured with a standard configuration file, which is supplied to the container through a SIP container custom property, as defined in JSR 289. The DAR configuration file can also be uploaded in the administrative console for each target of the DAR.

Application routing, also referred to as application composition, can be handled in a number of ways:

- Specify the order in which the applications should run using the administrative console.
- Upload a custom application router implementation class either by specifying the path of the Java archive (JAR) file containing the application router implementation and provider through the console or adding it to the class path. A specific provider can be defined with a SIP container custom property.
- Configure the DAR by uploading its properties file and providing its location through a system property.
- Use an interactive wizard to generate a DAR configuration file.

Restriction: WebSphere Application Server has a default way of sorting the order of SIP applications invocation using the Startup behavior settings. The sorting order is based on the application weight. This weighting policy only applies if you do not specify a DAR property file and no custom application router has been associated with the server or cluster.

Note: If CEA features are used, the CEA system application requires special consideration when enabled on the same server or cluster as a custom application router. To deploy an application router and still maintain the capabilities of the CEA system application, use one of the following two options:

- Only enable CEA on an isolated server or cluster that includes no custom application router.
- Make sure the custom application router routes all CEA specific messages to the CEA system application. To do this, the developer of the application router must check the mappings that are defined in the sip.xml file associated with the CEA system application. The sip.xml file associated with the CEA system application can be found in the directory path at *app_server_root/ systemApps*.

  The following information explains how to configure a custom application router to route to the commsvc system application. The examples show a custom application router configuration with and without the commsvc application.

  First, here is an example configuration without commsvc:

```
INVITE: ("TestB2bua", "DAR:To", "NEUTRAL", "", "NO_ROUTE", "0")
```

  The first element after the INVITE is the display name of the test application, and this one-line application router routes b2bua calls to the application successfully. With the preceding application router configured on the SIP container, however, CEA Web collaboration attempts fail.

  To enable routing to the CEA system application, just clone the routing element and change the application name in the second element instance:

```
INVITE: ("TestB2bua", "DAR:To", "NEUTRAL", "", "NO_ROUTE", "0"),("commsvc", "DAR:To", "NEUTRAL", "", "NO_ROUTE", "0")
```

  This action ensures that CEA messages are routed correctly.

Tuning considerations using the JSR 289 Application Router with multiple applications:

This topic describes performance adjustments and considerations using the JSR 289 Application Router™ with multiple applications.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

When you deploy more than one application, you might see the following errors in the log files when heavy SIP protocol traffic exists for a single application server or cluster of servers:

- Unexpected and excessive SIP application 503 Server Unavailable error messages
- Proxy and Server overload errors

Note: These error messages do not occur when you deploy one application.
The proxy server and Session Initiation Protocol (SIP) containers are not synchronized when they are tracking the amount of messages that are flowing through the system. Using the application router, multiple messages might be routed between applications. These messages cause container message counters to increment even though the messages do not flow through the proxy server.

You can diagnose this problem when you have the following conditions:

- Heavy SIP protocol traffic exists.
- Multiple applications are deployed on a single node or cluster.

Check the proxy server, the application server SystemOut.log log files, or both for an unexpected overload condition that is detected at the proxy server, the application server, or both. Also, look for 503 Server unavailable messages that are logged from the SIP application.

Resolving the problem

Messages are shared between applications at the SIP container before they are sent to the proxy server. To avoid these error messages and a decrease in SIP performance, tune the SIP containers to consider the additional SIP messages that are generated when using the application router with multiple applications. Complete the following steps in the administrative console to tune the SIP containers:

1. Expand **Servers** > **Server Types** and click **WebSphere application servers** > ***server_name***
2. Under **Container Settings**, expand **SIP Container Settings** and click **SIP container**.
3. Increase the **Maximum messages per averaging period** value to compensate for the anticipated increase in messages that are generated by the SIP application router.
4. Increase the **Maximum application sessions** value to compensate for the increased **Maximum messages per averaging period** value.

The proxy server cannot detect the amount of messages that are generated at the server. However, modifications to the following settings might increase the messaging capacity at the containers for the number of applications that are deployed per container.

Table 51. DAR and CAR SIP container tuning values.

This table lists the DAR and CAR SIP container tuning values for the number of applications that are deployed per container.

SIP Container	Single Deployed SIP Application	Three Deployed SIP Applications
Maximum messages per averaging period	value = 26640	value = 79920
Maximum application sessions	value = 36000	value = 96000

Note: The values for the **Maximum messages per averaging period** and **Maximum application sessions** fields depend on the processing power, memory, and the deployed application. Use the values for these fields as listed in the SIP container settings topic and adjust them to meet the needs of your environment.

SIP container

A *SIP container* is a web application server component that invokes the Session Initiation Protocol (SIP) action servlet and that interacts with the action servlet to process SIP requests.

The servlet container provides the network services over which requests and responses are received and sent. It decides which applications to invoke and in what order. The container also contains and manages servlets through their life cycle.

A SIP servlet container manages the network listener points on which it listens for incoming SIP traffic. A listener point is a combination of transport protocol, IP address, and port number. The SIP servlet container supports the transport protocols UDP, TCP, and TLS over TCP.

Chapter 22. Spring applications

This page provides a starting point for finding information about how to develop Spring applications that can run successfully in a WebSphere Application Server environment.

The Spring Framework is an open source project that provides a framework for simple Java objects that enables them to use the Java EE container through wrapper classes and XML configuration.

More introduction...

Spring Framework

There are some best practices to develop Spring Framework applications that can run successfully in a WebSphere Application Server environment.

The Spring Framework is an open source project that provides a framework for simple Java objects that enables them to use the Java EE container through wrapper classes and XML configuration.

You can use the Spring Framework with WebSphere Application Server Version 6.0.2 and later, but some supported features require a specific release of the product. When this situation applies, it is stated in the relevant topic.

For WebSphere Application Server Version 7.0 or later, you must use Spring Framework Version 2.5.5 or later.

In general, if both WebSphere Application Server and the Spring Framework provide a service, it is preferable to design your application to use the service in the application server directly. In this way, you ensure that the application is based on the open standards that the application server supports and has flexibility for future deployment. Also, you ensure that the application can use the qualities of service that the application server provides, such as security, workload management, and high availability.

Presentation layer and the Spring Framework

You can use the Spring Web model view controller (MVC) framework and the Spring Framework Portlet MVC framework with WebSphere Application Server.

Web MVC framework

WebSphere Application Server supports the use of the Spring Web MVC framework.

There are also Web MVC frameworks that are provided with WebSphere Application Server, such as JavaServer Faces (JSF) and Apache Struts, which have IBM product support. For information about how to integrate Spring with these Web MVC frameworks, see the Spring documentation.

Portlet MVC framework

The Spring Framework Portlet MVC framework can run in the portlet container in WebSphere Application Server Version 6.1 and later. To run portlets in the portlet container, you must create an additional web application to define the layout and aggregation of the portlets. For details about how to use the portlet aggregator tag library, see the related topics.

Data access and the Spring Framework

For Spring beans to access a data source, you must configure those beans so that the Spring Framework delegates to, and integrates with, the WebSphere Application Server runtime correctly.

The Spring Framework wraps Spring beans with a container-management layer that, in an enterprise application environment, delegates to the underlying enterprise application runtime. The following sections describe what to consider when you configure Spring beans that access a data source.

Access to data sources configured in the application server

For a Spring application to access a resource such as a Java Database Connectivity (JDBC) data source, the application must use a resource provider that is managed by the application server.

To do this, see the Configuring access to a Spring application data source topic.

JDBC native connections

WebSphere Application Server does not support the use of the NativeJdbcExtractor class that the Spring Framework provides, so avoid scenarios that use this class. Implementations of this class access native JDBC connections and bypass quality of service functions in the application server such as tracking and reassociating connection handles, sharing connections, managing connection pools and involvement in transactions.

As an alternative, you can use the application server WSCallHelper class to access non-standard vendor extensions for data sources.

Java Persistence API

WebSphere Application Server includes a default JPA provider based on the Apache OpenJPA implementation of JPA. For more information, see the related links.

To use the Spring Framework with a JPA implementation, it is advisable to use JPA directly rather than using the JPA helper classes that are provided with the Spring Framework in the org.springframework.orm.jpa package.

To use managed JPA from the Spring Framework, you define a persistence context reference in the web descriptor (`web.xml`):

```
<persistence-context-ref>
  <persistence-context-ref-name>some/name</persistence-context-ref-name>
  <persistence-unit-name>pu_name</persistence-unit-name>
</persistence-context-ref>
```

where *pu_name* is the name of the persistence unit as defined in the `persistence.xml` file.

The persistence context is then available from JNDI through `java:comp/env/some/name` inside the web application. For the Spring Framework, the persistence context can then be retrieved using a <jee:jndi-lookup/> as shown in the following example code. The resulting EntityManager object is available under the “entityManager” ID.

```
<jee:jndi-lookup id="entityManager" jndi-name="some/name" />
```

Similarly, a persistence unit (for direct use, or use with Spring wrapper classes) can be made available through a persistence unit reference:

```
<persistence-unit-ref>
  <persistence-unit-ref-name>some/ref_name</persistence-unit-ref-name>
  <persistence-unit-name>pu_name</persistence-unit-name>
</persistence-unit-ref>
```

The resulting EntityManagerFactory object is available under the “entityManagerFactory” ID:

```
<jee:jndi-lookup id="entityManagerFactory" jndi-name="some/ref_name" />
```

Transaction support and the Spring Framework

For Spring Framework Version 2.5 or later, you can use the declarative transaction model, use the Spring Framework support for the AspectJ programming extension, or use annotation-based transaction support. For versions of the Spring Framework earlier than Version 2.5, and for versions of the application server that do not provide the UOWManager interface, you can use a Spring Framework configuration that supports a restricted set of transaction attributes.

Declarative transaction model

WebSphere Application Server Version 6.0.2.19 or later and Version 6.1.0.9 or later support the Spring Framework declarative transaction model to drive resource updates under transactional control. The WebSphereUowTransactionManager class in Spring Framework 2.5 uses the UOWManager interface in the application server to manage the transaction context. Because transaction demarcation is managed through the UOWManager interface, an appropriate global transaction or local transaction containment (LTC) context is always available when a resource provider is accessed. For more information about the UOWManager interface and Java Transaction API (JTA) support, see the related topic.

The WebSphereUowTransactionManager class supports the following Spring Framework transaction attributes:

- PROPAGATION_REQUIRED
- PROPAGATION_SUPPORTS
- PROPAGATION_MANDATORY
- PROPAGATION_REQUIRES_NEW
- PROPAGATION_NOT_SUPPORTED
- PROPAGATION_NEVER

Use the following declaration for the WebSphere Application Server transaction support:

```
<bean id="transactionManager"
   class="org.springframework.transaction.jta.WebSphereUowTransactionManager"/>
```

A Spring bean that references the previous declaration can then use Spring Framework dependency injection to use the transaction support. For example:

```
<bean id="someBean" class="some.class">
   <property name="transactionManager" >
      <ref bean="transactionManager"/>
   </property>
...
</bean>
<property name="transactionAttributes">
   <props>
      <prop key="*">PROPAGATION_REQUIRED</prop>
   </props>
</property>
```

The AspectJ programming extension

You can use the Spring Framework support for the AspectJ programming extension. The following example code declares a <tx:advice/> element with the following transactional behavior:

- All methods that start with the string `get` have the transaction attribute PROPAGATION_REQUIRED.
- All methods that start with the string `set` have the transaction attribute PROPAGATION_REQUIRES_NEW.
- All other methods use the default transaction settings.

For example:

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="get*" propagation="REQUIRED" read-only="true" />
    <tx:method name="set*" propagation="REQUIRES_NEW" />
    <tx:method name="*" />
  </tx:attributes>
</tx:advice>
```

Then you can apply the settings to the required operation by declaring a pointcut. You can apply the settings to various parts of the application. The following example code applies the settings to any operation that is defined in the class MyService.

```
<aop:config>
  <aop:pointcut id="myServiceOperation"
                    expression="execution(* sample.service.MyService.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="myServiceOperation"/>
</aop:config>
```

Annotation-based transaction support

To use the annotation-based transaction support, you need Java Platform, Standard Edition 5 (Java SE 5) or later. Therefore, you can use this method with WebSphere Application Server Version 6.1 or later.

Add the following line to the Spring.xml configuration:

```
<tx:annotation-driven/>
```

Mark any methods that require transactional attributes with the `@Transactional` annotation, for example:

```
@Transactional(readOnly = true)
public String getUserName()
{ ...
}
```

You can use the `@Transactional` annotation to annotate only public methods.

Transaction support with Spring Framework before Version 2.5

You can use a Spring Framework configuration that supports a restricted set of transaction attributes.

You can use this method of transaction support with versions of the Spring Framework before Version 2.5 that do not provide the WebSphereUowTransactionManager class. You can also use this method of transaction support with versions of WebSphere Application Server earlier than Version 6.0.2.19 and Version 6.1.0.9 that do not provide the UOWManager interface.

The configuration supports the following Spring Framework transaction attributes:

- PROPAGATION_REQUIRED
- PROPAGATION_SUPPORTS
- PROPAGATION_MANDATORY
- PROPAGATION_NEVER

Use the following Spring Framework configuration:

```
<bean id="transactionManager"
             class="org.springframework.transaction.jta.JtaTransactionManager">
  <property name="autodetectTransactionManager"value="false" />
</bean>
```

The configuration does not support the following Spring Framework transaction attributes:

- PROPAGATION_REQUIRES_NEW
- PROPAGATION_NOT_SUPPORTED

WebSphere Application Server does not support the use of the Spring Framework class org.springframework.transaction.jta.WebSphereTransactionManagerFactoryBean.

JMX and MBeans with the Spring Framework

WebSphere Application Server Version 6.1 and later supports Spring Java Management Extensions (JMX) MBeans.

JMX and MBeans

To use the support for Spring JMX MBeans, you must register the JMX MBeans with the MBeanServer instance of the container manager in the application server. If you do not specify a server property for the MBean, the MBeanExporter object attempts to detect an MBeanServer instance that is running. Therefore, for an application that runs in the application server, the Spring Framework would locate the MBeanServer instance of the container.

Do not use the MBeanServerFactory class to instantiate an MBeanServer instance and then inject that instance into the MBeanExporter object. Also, do not use the Spring Framework ConnectorServerFactoryBean or JMXConnectorServer classes to expose the local MBeanServer instance to clients by opening inbound JMX ports.

Registering Spring MBeans in the application server

When an MBean is registered in the application server, it is identified by a fully qualified object name, javax.management.ObjectName. For example:

```
WebSphere:cell=99T73GDNode01Cell,name=JmxTestBean,node=99T73GDNode01,process=server1,
type=JmxTestBeanImpl
```

When an MBean is deregistered, it must looked up using the same fully qualified name, rather than just the name property of the MBean. The best way to manage this to implement the org.springframework.jmx.export.naming.ObjectNamingStrategy interface. The ObjectNamingStrategy interface encapsulates the creation of ObjectName objects, and is used by the MBeanExporter class to obtain ObjectNames when beans are registered. You can add the ObjectNamingStrategy instance to the bean that you register so that the MBean is deregistered properly when the application is uninstalled. For example:

```
<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter"
  lazy-init="false">
  <property name="beans">
    <map> <entry key="JmxTestBean" value-ref="testBean" /> </map>
  </property>
  <property name="namingStrategy" ref="websphereNamingStrategy" />
...
</bean>
```

MBeans and notifications

To use notifications, it is advisable to define the object name for an MBean in full, because the MBean is identified by a fully qualified object name when it is registered in WebSphere Application Server. For example:

```
<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter"
  lazy-init="false">
  <property name="beans">
    <map>
      <entry key="JmxTestBean" value-ref="testBean" />
    </map>
  </property>
  <property name="namingStrategy" ref="websphereNamingStrategy" />
  <property name="notificationListenerMappings">
    <map>
      <entry key="WebSphere:cell=99T73GDNode01Cell, name=JmxTestBean,
   node=99T73GDNode01, process=server1, type=JmxTestBeanImpl">
        <bean class="client.MBeanListener" />
      </entry>
    </map>
  </property>
</bean>
```

JMS and the Spring Framework

A Spring Framework application can use the JMSTemplate class to send JMS messages or receive synchronous JMS messages.

The JMSTemplate can locate JMS destinations from their Java Naming and Directory Interface (JNDI) name that you configure in an application resource reference.

Alternatively, for Spring Framework Version 2.5 and later, the JMSTemplate can locate JMS destinations through dynamic resolution, which looks up the administrative name of the destination that is configured in WebSphere Application Server

You use a Spring JndiObjectFactoryBean as a proxy for a ConnectionFactory to ensure that JMS resources are managed correctly. For example:

```
<bean id="jmsConnectionFactory"
   class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="jndiName" value="java:comp/env/jms/myCF"/>
      <property name="lookupOnStartup" value="false"/>
      <property name="cache" value="true"/>
      <property name="proxyInterface" value="javax.jms.ConnectionFactory"/>
</bean>
```

The following example shows the configuration of a resource reference for a ConnectionFactory. During application deployment, this resource reference is mapped to a configured, managed Connection Factory that is stored in the JNDI namespace of the application server. The ConnectionFactory is required to undertake messaging and should be injected into the Spring JMSTemplate object.

```
<resource-ref>
  <res-ref-name>jms/myCF</res-ref-name>
  <res-type>javax.jms.ConnectionFactory</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

After there is a defined JNDI name for the ConnectionFactory in the application, that JNDI name can be looked up and injected into the JMSTemplate. For example:

```
<jee:jndi-lookup id="jmsConnectionFactory" jndi-name=" jms/myCF "/>

<bean id="jmsQueueTemplate" class="org.springframework.jms.core.JmsTemplate">
  <property name="connectionFactory">
    <ref bean="jmsConnectionFactory"/>
  </property>
  <property name="destinationResolver">
    <ref bean="jmsDestResolver"/>
  </property>

  ...
</bean>

<!-- A dynamic resolver -->
<bean id="jmsDestResolver"
  class=" org.springframework.jms.support.destination.DynamicDestinationResolver"/>

<!-- A JNDI resolver -->
<bean id="jmsDestResolver"
  class=" org.springframework.jms.support.destination.JndiDestinationResolver"/>
```

At run time, the JMSTemplate object can locate a destination based on its JNDI name that was configured in an application resource reference. Alternatively, the JMSTemplate object can locate a destination by using dynamic resolution, based on the administrative name of the destination configured in WebSphere Application Server.

The following example shows how to use JNDI resolution to locate the JMS queue myQueue that is bound to a JNDI reference of jms/myQueue:

```
jmsTemplate.send("java:comp/env/jms/myQueue", messageCreator);
```

The following example shows how to use dynamic resolution to locate the JMS queue myQueue that is bound to a JNDI reference of jms/myQueue:

```
jmsTemplate.send("myQueue", messageCreator);
```

Class loaders and the Spring Framework

You can avoid potential problems with the class loading of Java archive (JAR) files and resources.

If there are problems with the class loading of JAR files and resources, there can be exceptions in the log about version mismatches of classes, ClassCastException exceptions, or java.lang.VerifyError exceptions. To avoid class loading problems, ensure that Spring Framework dependencies are packaged as part of the application, and configure the class loader policy of the server so that the application server run time uses the version that you intend. For example, consider changing the search order in the class loader configuration to “parent last”.

It is possible for resources that use a common name to be found in an unintended location. Resources can include message bundles. You can use the class loader viewer in the application server to help resolve this problem. You might want the application to rename resources so that they have a unique name. For more information about the class loader viewer, see the topic about troubleshooting class loaders.

Thread management and the Spring Framework

Use the information in the following sections to avoid potential problems with unmanaged threads.

Unmanaged threads

Do not use a scenario that can create unmanaged threads, for the following reasons:

- The application server does not recognize unmanaged threads.
- Unmanaged threads do not have access to Java EE contextual information.
- Unmanaged threads can use resources without being monitored by the application server.
- Unmanaged threads can adversely affect application server functions such as shutting down gracefully or recovering resources from failure.
- An administrator cannot control the number of unmanaged threads or their use of resources.

The following scenarios are examples of Spring Framework scenarios to avoid:

- registerShutdownHook

  Avoid using the Spring Framework AbstractApplicationContext class and its subclasses. These classes include the public method registerShutdownHook, which creates a thread and registers it with the Java virtual machine (JVM) to run at shutdown to close the application context. As an alternative, an application can use the lifecycle notices that it receives from the application server container to call the close method explicitly on the application context.
- WeakReferenceMonitor

  The Spring Framework provides convenience classes for simplified development of EJB components. However, these convenience classes spawn off an unmanaged thread that WeakReferenceMonitor object uses for cleanup.

Thread pooling

WebSphere Application Server supports the use of the Spring Framework WorkManagerTaskExecutor class to run work asynchronously.

The WorkManagerTaskExecutor class uses thread pools that are managed by the application server, and delegates to a configured WorkManager instance. For information about configuring a work manager, see the related topics.

Do not use other TaskExecutor classes that are provided with the Spring Framework, because they might start unmanaged threads.

You can use the Java Naming and Directory Interface (JNDI) name of the configured work manager as a workManagerName property to define a WorkManagerTaskExecutor instance in the Spring configuration file. The following example uses the JNDI name of the DefaultWorkManager in the application server, that is, `wm/default`:

```
<bean id="myTaskExecutor"
  class="org.springframework.scheduling.commonj.WorkManagerTaskExecutor">
  <property name="workManagerName" value="wm/default" />
</bean>
```

Scheduling

You can use the CommonJ WorkManager scheduling package in the Spring Framework to work with threads that are managed by the application server. Avoid using other packages, such as the Quartz scheduler, or the Timer in the Java SE Development Kit (JDK), because they can start unmanaged threads.

Chapter 23. Transactions

This page provides a starting point for finding information about Java Transaction API (JTA) support. Applications running on the server can use transactions to coordinate multiple updates to resources as one unit of work, such that all or none of the updates are made permanent.

The product provides advanced transactional capabilities to help application developers avoid custom coding. It provides support for the many challenges related to integrating existing software assets with a Java EE environment.

More introduction...

Transaction support in WebSphere Application Server

Support for transactions is provided by the transaction service within WebSphere Application Server. The way that applications use transactions depends on the type of application component.

A transaction is unit of activity, within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, during the processing of an SQL COMMIT statement, the database manager atomically commits multiple SQL statements to a relational database. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a *logical unit of work* (LUW). If a transaction involves multiple resource managers, for example multiple database managers, an external transaction manager is required to coordinate the individual resource managers. A transaction that spans multiple resource managers is referred to as a *global transaction*. WebSphere Application Server is a transaction manager that can coordinate global transactions, can be a participant in a received global transaction, and can also provide an environment in which resource manager local transactions can run.

The way that applications use transactions depends on the type of application component, as follows:
- A session bean can use either container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (component-managed transactions where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) and application client components use component-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface, and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service (OTS) protocol or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through Java EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with databases, JMS queues, and JCA connectors through their *local transaction* support, when you do not require distributed transaction coordination.

IBM i Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances, you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:

- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to complete any updates, the one-phase commit resource does not have to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one or more two-phase commit resource providers and where *last participant support* is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the Java EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:

- Enterprise Information Systems that are accessed through a resource adapter, as described in the Java EE Connector Architecture.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. Resource adapter components of the Java EE connector architecture that include support for local transactions provide a LocalTransaction interface. The LocalTransaction interface enables applications to request that the resource adapter commits or rolls back RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, an external transaction manager is needed to coordinate the updates to all the resource managers in a global transaction.

Global transaction support is available to web and enterprise bean components and, with some limitations, to application client components. Enterprise bean components can be subdivided into two categories: beans that use container-managed transactions (CMT) and beans that use bean-managed transactions (BMT).

Only BMT enterprise beans, application client components, and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. To obtain the UserTransaction interface, use a Java Naming and Directory Interface (JNDI) lookup of java:comp/UserTransaction, or use the getUserTransaction method from the SessionContext object.

The UserTransaction interface is not available to CMT enterprise beans. If CMT enterprise beans attempt to obtain this interface, an exception is thrown, in accordance with the Enterprise JavaBeans (EJB) specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location that is appropriate for the EJB version.

WebSphere Application Server Version 4 and later releases bind the UserTransaction interface at the JNDI location that is specified in the EJB Version 1.1 specification. This location is java:comp/UserTransaction.

A web component or enterprise bean (CMT or BMT) can use additional interfaces that provide JTA support. These interfaces provide the transaction identity and a mechanism to receive notification of transaction completion. The interfaces include the TransactionSynchronizationRegistry interface, the ExtendedJTATransaction interface, and the UOWSynchronizationRegistry interface.

Local transaction containment

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

Unspecified transaction context is defined in the Enterprise JavaBeans specification, Version 2.0 and later. For example, see the specification for this technology.

An LTC is a bounded unit-of-work scope, within which zero or more resource manager local transactions (RMLT) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. By default, an LTC is local to a bean instance; it is not shared across beans, even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on an enterprise application component, such as an enterprise bean or servlet, whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example, at the end of the method dispatch. There is no programmatic interface to the LTC support; LTCs are managed exclusively by the container. The application deployer configures LTCs on individual application components, either web application or EJB, by using transaction attributes in the application deployment descriptor.

A local transaction containment (LTC) might be configured as part of an application component's deployment descriptor to be shareable across multiple application components, including web application components and enterprise beans that use container-managed transactions, so that those components can share connections without using a global transaction. Sharing a single resource manager between application components improves performance, increases scalability, and reduces lock contention for resources.

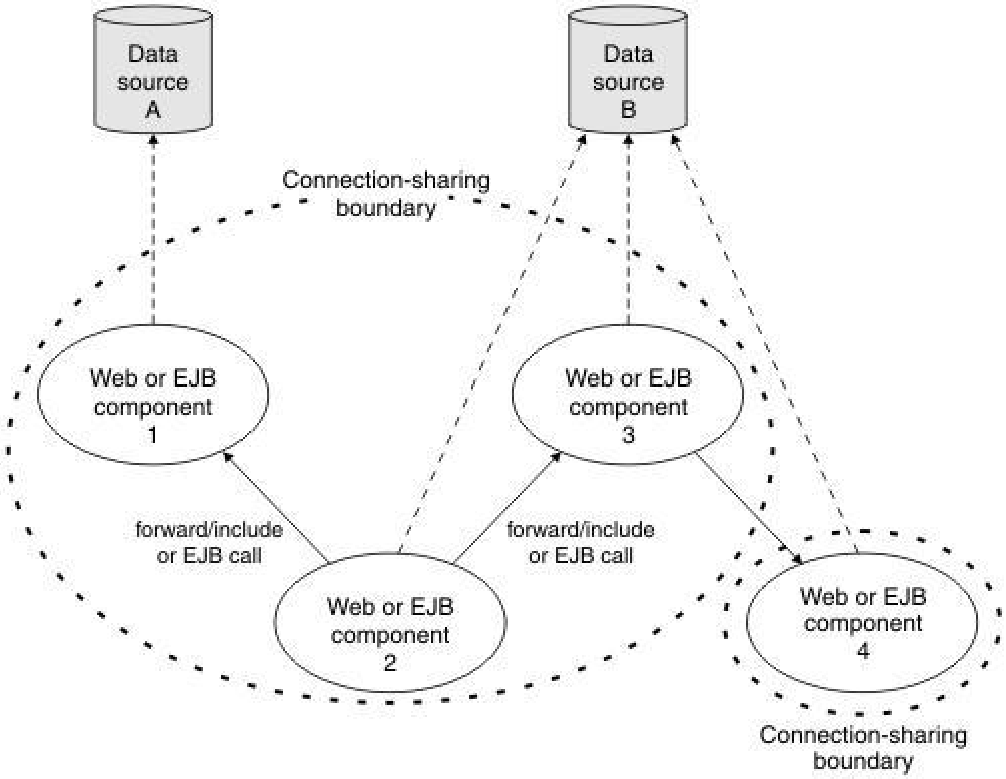
LTCs can be shared across multiple components, including web application components and enterprise beans that use container-managed transactions. This sharing is useful in situations such as frequent use of web component `include()` calls, where a thread can have several connections blocked by LTCs in different web modules. In this situation, the application might encounter code deadlocks under load, when threads start to wait for themselves to free connections. To overcome this issue without using a global transaction, specify that application components can share LTCs by setting the Shareable attribute in the deployment descriptor of each component. You must use a deployment descriptor; you cannot specify this attribute if annotation has been used.

When you set the Shareable attribute, the extended deployment descriptor XML file includes the following line of code:

```
<local-transaction boundary="BEAN_METHOD" resolver="CONTAINER_AT_BOUNDARY"
unresolved-action="COMMIT" shareable="true"/>
```

To obtain the full benefits of a shared LTC, also ensure that the resource reference for each component defaults to shareable connections.

In the following diagram, components 1, 2 and 3 are deployed with the Shareable attribute and component 4 is not. If components 2 and 3 both obtain connections to data source B, and their resource references for data source B default to shareable connections, they share the connection, but component 4 does not.



Applications that use shareable LTCs cannot explicitly commit or roll back resource manager connections that are used in a shareable LTC. Although, they can use connections that have an autoCommit capability. This ensures correct encapsulation of connection usage by each component and protects one component from having to make any assumptions about the behavior of other components that share the connection.

If an application starts any non-autocommit work in an LTC for which the Resolver attribute is set to `Application` and the Shareable attribute is set to `true`, an exception occurs at run time. For example, on a JDBC connection, non-autocommit work is work that the application performs after using the setAutoCommit(false) method to disable the autocommit option on the connection. Enterprise beans that use bean managed transactions (BMT) cannot be assembled with the Shareable attribute set on the LTC configuration.

IBM i A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for enterprise application components at J2EE 1.3 or later. The only exceptions to this are the following items:

- Application component dispatch occurs without container interposition, for example, for a stateless session bean create method or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 bean-managed transaction (BMT) enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that exists longer than the enterprise bean method in which it is started, as described in the topic about ActivitySessions and transaction contexts.

Local transaction containment

IBM WebSphere Application Server supports local transaction containment (LTC), which you can configure using local transaction extended deployment descriptors. LTC support provides certain advantages to application programmers. Use the scenarios provided, and the list of points to consider, to help you decide the best way to configure transaction support for local transactions.

The following sections describe the advantages that LTC support provides, and how to set the local transaction extended deployment descriptors in each situation.

You can develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not have to use global transactions, it is often more efficient to deploy the bean with the deployment descriptor for the container transaction type set to NotSupported instead of Required.

With the extended local transaction support of the application server, applications can perform the same business logic in an unspecific transaction context as they can in a global transaction. An enterprise bean, for example, runs in an unspecified transaction context if it is deployed with a container transaction type of NotSupported or Never.

The extended local transaction support provides a container-managed, implicit local transaction boundary, within which the container commits application updates and cleans up their connections. You can design applications with more independence from deployment concerns. This makes using a container transaction type of Supports much simpler, for example, when the business logic might be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage, regardless of whether the application runs in a transaction. The application can depend on the close action behaving in the same way in all situations, that is, the close action does not cause a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios, running business logic in a Transaction policy of NotSupported performs better than in a policy of Required. This benefit is applied through setting the deployment descriptor, in the Local Transactions section, of the Resolver attribute to ContainerAtBoundary. With this setting, application interactions with resource providers, such as databases, are managed within implicit resource manager local transactions (RMLT) that the container both starts and ends. The container commits RMLTs at the containment boundary that is specified by the Boundary attribute in the Local Transactions section; for example, at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

You can use local transactions in a managed environment that guarantees cleanup.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default setting of Application for the Resolver extended deployment descriptor in the Local Transactions section. In this situation, the container ensures connection cleanup at the boundary of the local transaction context.

Java platform for enterprise applications specifications that describe application use of local transactions do so in the manner provided by the default settings of Application for the Resolver extended deployment descriptor, and Rollback for the Unresolved action extended deployment descriptor, in the Local Transactions section. When the Unresolved action extended deployment descriptor in the Local Transactions section is set to Commit, the container commits any RMLTs that the application starts but that do not complete when the local transaction containment ends (for example, when the method ends). This usage applies to both servlets and enterprise beans.

IBM i **You can extend the duration of a local transaction beyond the duration of an EJB component method.**

The Enterprise JavaBeans (EJB) specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can use the Boundary extended deployment setting in the Local Transactions section to give the following advantages:

- Significantly extend the use cases of RMLTs.
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which a client can control. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is applied through the Boundary extended deployment setting, in the Local transactions section, of ActivitySession. Such extended RMLTs can remain under the control of the application, or be managed by the container, depending on the setting of the Resolver deployment descriptor in the Local Transactions section.

You can coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can use ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans in that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

You can use shareable LTCs to reduce the number of connections you require.

Application components can share LTCs. If components obtain connections to the same resource manager, they can share that connection if they run under the same global transaction or shareable LTC. To configure two components to run under the same shareable LTC, set the Shareable attribute of the Local Transactions section in the deployment descriptor of each component. Make sure that the resource reference in the deployment descriptor for each component uses the default value of Shareable for the res-sharing-scope element, if this element is specified. A shareable LTC can reduce the numbers of RMLTs an application uses. For example, an application that makes frequent use of web module include calls can share resource manager connections between those web modules, exploiting either shareable LTCs, or a global transaction, reducing lock contention for resources.

Examples of local transaction support configurations

The following list gives scenarios that use local transactions, and points to consider when deciding the best way to configure the transaction support for an application.

- You want to start and end global transactions explicitly in the application (bean-managed transaction session beans and servlets only).

  For a session bean, set the Transaction type to Bean (to use bean-managed transactions) in the deployment descriptor of the component. You do not have to do this for servlets.
- IBM i You want to access only one XA or non-XA resource in a method.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to Supports.
- You want to access several XA resources atomically across one or more bean methods.

In the deployment descriptor of the component, in the Container Transactions section, set the container transaction type to Required, RequiresNew, or Mandatory.

- IBM i You want to access several non-XA resources in a method without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to NotSupported.
- You want to access several non-XA resources in a method and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Unresolved action attribute to Rollback. In the Container Transactions section, set the container transaction type to NotSupported.
- IBM i You want to access one or more non-XA resources across multiple EJB method calls without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.
- IBM i You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.

Local and global transactions

Applications use resources, such as Java Database Connectivity (JDBC) data sources or connection factories, that are configured through the Resources view of the administrative console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider.

For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLT), but an XA data source can support two-phase commit coordination, as well as local transactions.

IBM i If an application uses two or more resource providers that support only RMLTs, atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application must use resources that support XA coordination and must access those resources in a global transaction.

If an application uses only one RMLT, atomic behavior can be guaranteed by the resource manager, which can be accessed in a local transaction containment (LTC) context.

An application can also access a single resource manager in a global transaction context, even if that resource manager does not support the XA coordination. An application can do this because the application server performs an “only resource optimization” and interacts with the resource manager in a RMLT. In a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but not both. An instance of an enterprise bean can change from running in one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions

Application clients can, within certain limits, support the use of transactions.

Application clients running in an enterprise application client container can explicitly demarcate transaction boundaries, as described in the topic about using component-managed transactions. Application clients cannot perform, directly in the client container, transactional work in the context of any global transaction that they start, because the client container is not a recoverable process.

Application clients can make requests to remote objects, such as enterprise beans, in the context of a client-initiated transaction. Any transactional work performed in a remote, recoverable, server process is coordinated as part of the client-initiated transaction. The transaction coordinator is created on the first server process to which the client-initiated transaction is propagated.

A client can begin a transaction, then, for example, access a JDBC data source directly in the client process. In such cases, any work performed through the JDBC provider is not coordinated as part of the global transaction. Instead, the work runs under a resource manager local transaction. The client container process is non-recoverable and contains no transaction coordinator with which a resource manager can be enlisted.

A client can begin a transaction, then call a remote application component such as an enterprise bean. In such cases, the client-initiated transaction context is implicitly propagated to the remote application server, where a transaction coordinator is created. Any resource managers accessed on the recoverable application server (or any other application server hosting application components invoked by the client) are enlisted in the global transaction.

Client application components must be aware that locally-accessed resource managers are not coordinated by client-initiated transactions. Client applications acknowledge this through a deployment option that enables access to the UserTransaction interface in the client container. By default, access to the UserTransaction interface in the client container is not enabled. To enable UserTransaction demarcation for an application client component, set the “Allow JTA Demarcation” extension property in the client deployment descriptor. For information about editing the client deployment descriptor, refer to the Rational Application Developer information.

Commit priority for transactional resources

You can specify the order in which transactional resources are processed during two-phase commit processing.

If you control the order in which transactional resources are processed during two-phase commit processing, there are two main benefits:

- One-phase commit optimization occurs more often.
- Potential problems caused by transaction isolation are resolved.

To control the order in which transactional resources are processed during two-phase commit processing, you specify the commit priority of a resource by setting the commit priority attribute on a resource reference. The larger the commit priority, the earlier the resource is processed. For example, if a resource has a commit priority of 10, it is processed before a resource with a commit priority of 1. The commit priority value is of type int and can be between -2147483648 and 2147483647.

If you do not specify a commit priority value, a default value of zero is assigned to the resource and is used when ordering resources at run time. If two or more resources are configured with the same priority, including the default priority, they are processed in an unspecified order with respect to each other.

You can specify the commit priority attribute on a resource reference by using Rational Application Developer tools. For detailed information, see the Rational Application Developer information center. The application component must have a deployment descriptor; you cannot specify this attribute if annotation has been used.

One-phase commit optimization

In a transaction with a two-phase commit, if every resource except the last one enlisted in the transaction votes read-only, indicating that those resources are not interested in the outcome of the transaction, a one-phase commit can occur. This means that the transaction service does not have to store resource and transaction information that it would need to roll back a two-phase commit, and therefore performance is improved.

You can control the order in which transactional resources are processed during two-phase commit, so you can process the resources that are most likely to vote read-only first. Therefore, you increase the chance that a one-phase commit might occur.

Typically, for a given transactional resource, you know the work that is performed at run time, so if you can control the order in which the resources in a transaction are processed, you can increase the likelihood of a one-phase commit optimization occurring.

Transaction isolation

When resources are involved in a global transaction, updates that are made as part of a transaction are not visible outside the transaction until the transaction commits, that is, those resources are isolated. This isolation can cause problems with other application components that act on the updates after they are committed. For example, further processing can fail, or can fail intermittently, because updates are order and time dependent. This problem does not occur with service integration bus messaging work in WebSphere Application Server, but can be a problem for other messaging providers, for example WebSphere MQ.

If you specify the order in which transactional resources are committed, problems caused by isolation are resolved for all transactional systems, not just messaging providers and service integration bus in particular.

The following example describes how problems might occur when you cannot specify the order in which transactional resources are committed. An application updates a row in a database table, then sends a JMS message that triggers additional processing of the row. Both of these actions are performed in the same global transaction, so they are isolated until their respective resources are committed. If the update to the row is committed before the message is sent, the processing that is triggered by the message can access the updated row and process it. If the action to send the message is committed first, this action might trigger the additional processing of the row before the database has committed the update to the row. In this situation, the updated row is still isolated and is not visible, so the additional processing of the row fails.

This problem can be more complicated because it is ordering and timing dependent. If the database is committed first, the problem does not occur. If the action to send the message is committed first, the problem might occur, but it depends whether the database work is committed before the message triggers the further processing of the row. Therefore, the problem can be intermittent, so it is harder to identify its cause.

Restrictions with earlier versions of WebSphere Application Server

If you specify the commit priority of a resource, that is, specify any value other than the default value 0, the commit priority is added to the partner log in a recoverable unit section. This section in the log file is recognized in WebSphere Application Server Version 7.0 or later, but not in earlier versions of the application server.

Therefore, if an application uses the commit priority attribute, you cannot install that application into a mixed-version cluster where one or more servers in the cluster are at versions of WebSphere Application Server that are earlier than Version 7.0.

Also, if an application that uses the commit priority attribute is installed in a cluster, you cannot subsequently add a server to that cluster if the server is at a version of WebSphere Application Server that is earlier than Version 7.0.

For general information about different versions of the product, see the topic “Overview of migration, coexistence, and interoperability”.

Transaction compensation and business activity support

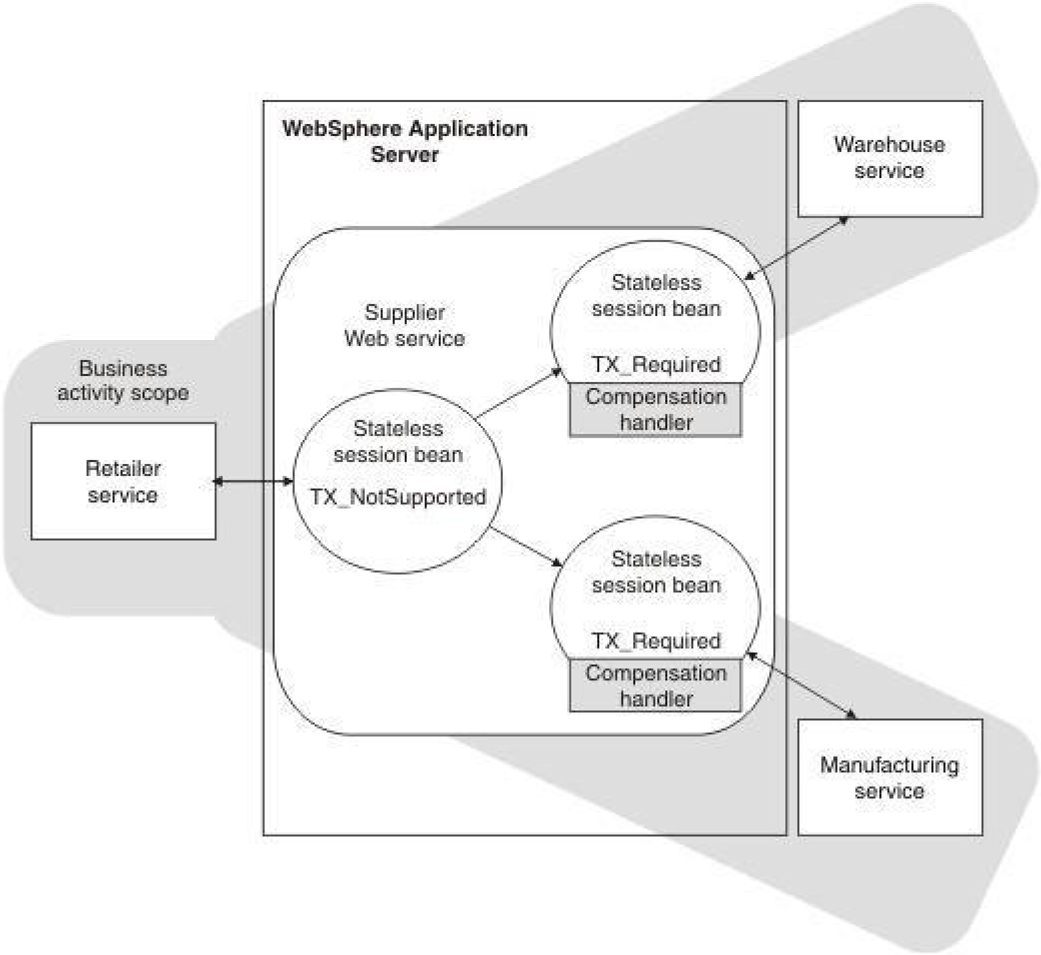
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an email can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Serverenvironments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that undertake work that can be compensated. These other session beans each have a *compensation handler*, a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an email.



Application design

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work by using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functions. Enterprise beans that exploit business activity scopes can offer web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in web service messages by using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

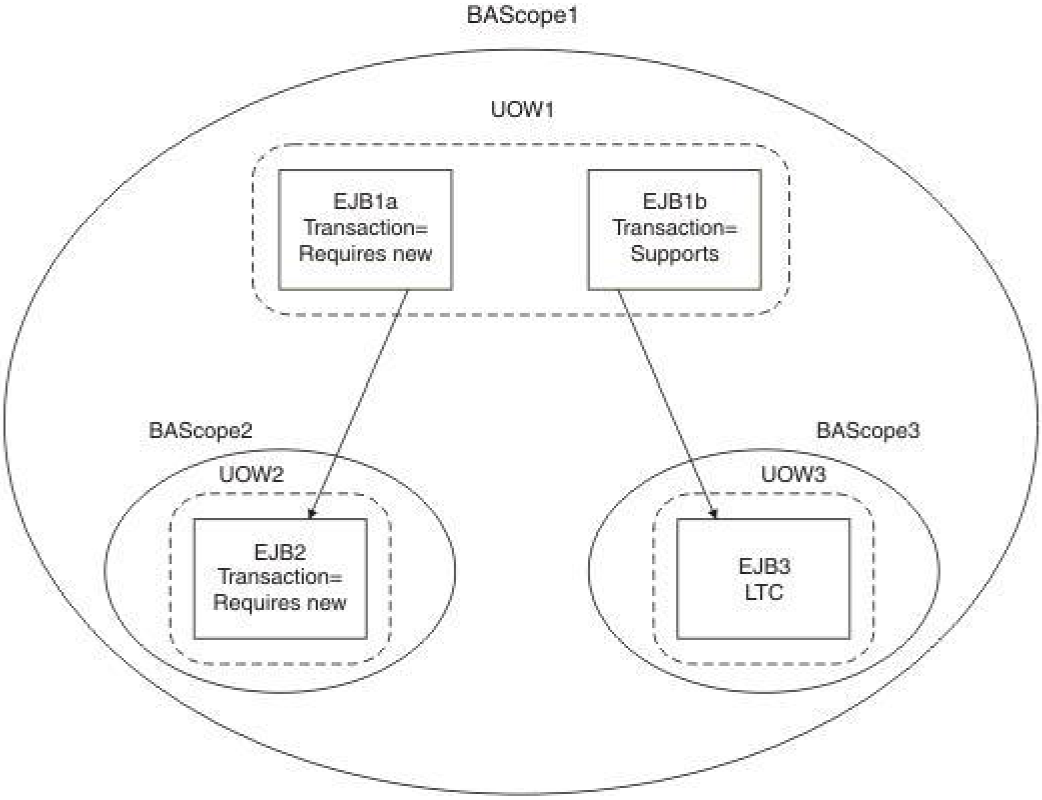
Business activity scopes

The scope of a business activity is that of a main WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing main UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any main UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.



Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by trying the called component again or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 52. Compensation behavior for a single business activity scope. The table lists the possible combinations of success and failure for the inner and outer business activity scopes, and the compensation behavior associated with each combination.

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see the topic about the business activity API.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight by using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration

If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope

Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the

received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method, then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For details, see the topic about creating an application that uses the WS-BA support.

JTA support

Java Transaction API (JTA) support provides application programming interfaces (APIs) in addition to the UserTransaction interface that is defined in the JTA 1.1 specification.

These interfaces include the TransactionSynchronizationRegistry interface, which is defined in the JTA 1.1 specification, and the following API extensions:
- SynchronizationCallback interface
- ExtendedJTATransaction interface
- UOWSynchronizationRegistry interface
- UOWManager interface

The APIs provide the following functions:
- Access to global and local transaction identifiers associated with the thread.

  The global identifier is based on the transaction identifier in the CosTransactions::PropagationContext: object and the local identifier identifies the transaction uniquely in the local Java virtual machine (JVM).
- A transaction synchronization callback that any enterprise application component can use to register an interest in transaction completion.

  Advanced applications can use this callback to flush updates before transaction completion and clear up state after transaction completion. Java EE (and related) specifications position this function typically as the domain of the enterprise application containers.

  Components such as persistence managers, resource adapters, enterprise beans, and web application components can register with a JTA transaction.

The following information is an overview of the interfaces that the JTA support provides. For more detailed information, see the generated API documentation.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run on a Java platform for enterprise applications server, there is no specific enterprise application component active when this object is called. So, the object has limited direct access to any enterprise application resources. Specifically, the object has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to an enterprise application component (for example, a stateless session bean) that it delegates to. The object would then have all the usual access to enterprise application resources. For example, you might use the object to acquire a Java Database Connectivity (JDBC) connection and flush updates to a database during the beforeCompletion method.

ExtendedJTATransaction interface

This interface is a WebSphere programming model extension to the Java EE JTA support. An object implementing this interface is bound, by enterprise application containers in WebSphere Application Server

that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an Enterprise JavaBeans (EJB) container, is not restricted to component-managed transactions.

An application uses a Java Naming and Directory Interface (JNDI) lookup of java:comp/websphere/ ExtendedJTATransaction to get an ExtendedJTATransaction object, which the application uses as shown in the following example:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallback objects. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server, whether the transaction is started locally or imported
- At the end of the transaction for which the callback was registered

Note: In this release, the registerSynchronizationCallbackForCurrentTran method is deprecated. Use the registerInterposedSynchronization method of the TransactionSynchronizationRegistry interface instead.

TransactionSynchronizationRegistry interface

This interface is defined in the JTA 1.1 specification. System-level application components, such as persistence managers, resource adapters, enterprise beans, and web application components, can use this interface to register with a JTA transaction. Then, for example, the component can flush a cache when a transaction completes.

To obtain the TransactionSynchronizationRegistry interface, use a JNDI lookup of java:comp/ TransactionSynchronizationRegistry.

Note: Use the registerInterposedSynchronization method to register a synchronization instance, rather than the registerSynchronizationCallbackForCurrentTran method of the ExtendedJTATransaction interface, which is deprecated in this release.

UOWSynchronizationRegistry interface

This interface provides the same functions as the TransactionSynchronizationRegistry interface, but applies to all types of units of work (UOWs) that WebSphere Application Server supports:

- JTA transactions
- local transaction containments (LTCs)
- ActivitySession contexts

System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface to register with a JTA transaction. The component can do the following:

- Register synchronization objects with special ordering semantics.
- Associate resource objects with the UOW.
- Get the context of the current UOW.
- Get the current UOW status.
- Mark the current UOW for rollback.

To obtain the UOWSynchronizationRegistry interface, use a JNDI lookup of java:comp/websphere/UOWSynchronizationRegistry. This interface is available only in a server environment.

The following example registers an interposed synchronization with the current UOW:

```
// Retrieve an instance of the UOWSynchronizationRegistry interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWSynchronizationRegistry uowSyncRegistry =
  (UOWSynchronizationRegistry)initialContext.lookup("java:comp/websphere/UOWSynchronizationRegistry");

// Instantiate a class that implements the javax.transaction.Synchronization interface
final Synchronization sync = new SynchronizationImpl();

// Register the Synchronization object with the current UOW.
uowSynchronizationRegistry.registerInterposedSynchronization(sync);
```

UOWManager interface

The UOWManager interface is equivalent to the JTA TransactionManager interface, which defines the methods that allow an application server to manage transaction boundaries. Applications can use the UOWManager interface to manipulate UOW contexts in the product. The UOWManager interface applies to all types of UOWs that WebSphere Application Server supports; that is, JTA transactions, local transaction containments (LTCs), and ActivitySession contexts. Application code can run in a particular type of UOW without needing to use an appropriately configured enterprise bean. Typically, the logic that is performed in the scope of the UOW is encapsulated in an anonymous inner class. System-level application server components such as persistence managers, resource adapters, enterprise beans, and web application components can use this interface.

WebSphere Application Server does not provide a TransactionManager interface in the API or the system programming interface (SPI). The UOWManager interface provides equivalent functions, but WebSphere Application Server maintains control and integrity of the UOW contexts.

To obtain the UOWManager interface in a container-managed environment, use a JNDI lookup of java:comp/websphere/UOWManager. To obtain the UOWManager interface outside a container-managed environment, use the UOWManagerFactory class. This interface is available only in a server environment.

You can use the UOWManager interface to migrate a web application to use web components rather than enterprise beans, but maintain control over the UOWs. For example, a web application currently uses the UserTransaction interface to begin a global transaction, makes a call to a method on a session enterprise bean that is configured as not supported to undertake some non-transactional work, and then completes the global transaction. You can move the logic that is encapsulated in the session EJB method to the run method of a UOWAction implementation. Then, you replace the code in the web component that calls the session enterprise bean with a call to the runUnderUOW method of a UOWManager interface to request that this logic is run in a local transaction. In this way, you maintain the same level of control over the UOWs as you had with the original application.

The following example performs some transactional work in the scope of a new global transaction. The transactional work is performed in an anonymous inner-class that implements the run method of the UOWAction interface. Any checked exceptions that the run method creates do not affect the outcome of the transaction.

```
// Retrieve an instance of the UOWManager interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWManager uowManager = (UOWManager)initialContext.lookup("java:comp/websphere/UOWManager");

try
{
 // Invoke the runUnderUOW method, indicating that the logic should be run in a global
 // transaction, and that any existing global transaction should not be joined, that is,
 // the work must be performed in the scope of a new global transaction.
 uowManager.runUnderUOW(UOWSynchronizationRegistry.UOW_TYPE_GLOBAL_TRANSACTION, false, new UOWAction()
 {
   public void run() throws Exception
   {
```

```
   // Perform transactional work here.
   }
 });
}

catch (UOWActionException uowae)
{
 // Transactional work resulted in a checked exception being thrown.
}

catch (UOWException uowe)
{
 // The completion of the UOW failed unexpectedly. Use the getCause method of the
 // UOWException to retrieve the cause of the failure.
}
```

Chapter 24. Work area

This page provides a starting point for finding information about work areas, a WebSphere extension for improving developer productivity.

Work areas provide a capability much like that of global variables. They enable efficient sharing of information across a distributed application.

For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it is available throughout your application, eliminating the need to hand-code a solution or to read and write information to a database.

Overview of work area service

The work area service passes information explicitly as an argument or implicitly to remote methods.

One of the foundations of distributed computing is the ability to pass information, typically in the form of arguments to remote methods, from one process to another. When application-level software is written over middleware services, many of the services rely on information beyond that passed in the application's remote calls. Such services often make use of the implicit propagation of private information in addition to the arguments passed in remote requests; two typical users of such a feature are security and transaction services. Security certificates or transaction contexts are passed without the knowledge or intervention of the user or application developer. The implicit propagation of such information means that application developers do not have to manually pass the information in method invocations, which makes development less error-prone, and the services requiring the information do not have to expose it to application developers. Information such as security credentials can remain secret.

The work area service gives application developers a similar facility. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of every method. The methods on the server side can use or ignore the information in the work area as appropriate. If methods in a server receive a work area from a client and subsequently invoke other remote methods, the work area is transparently propagated with the remote requests. When the creating application is done with the work area, it terminates it.

There are two prime considerations in deciding whether to pass information explicitly as an argument or implicitly by using a work area. These considerations are:

- Pervasiveness: Is the information used in a majority of the methods in an application?
- Size: Is it reasonable to send the information even when it is not used?

When information is sufficiently pervasive that it is easiest and most efficient to make it available everywhere, application programmers can use the work area service to simplify programming and maintenance of code. The argument does not need to go onto every argument list. It is much easier to put the value into a work area and propagate it automatically. This is especially true for methods that simply pass the value on but do nothing with it. Methods that make no use of the propagated information simply ignore it.

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data, each stored as a property.

Use the work area service in the administrative console to configure the UserWorkArea partition. The UserWorkArea partition is the partition that is available in JNDI naming under the name "java:comp/websphere/UserWorkArea", as demonstrated in the Accessing the UserWorkArea partition article. The UserWorkArea partition is the default work area partition created automatically, if it has not been disabled, and is available through JNDI naming to all users. Any configuration option made to the

UserWorkArea partition under the work area service panel in the administrative console does not affect the work area partition service or any partitions defined in it, and conversely. For example, if you select the enable or disable option in the work area service panel, this does not affect the work area partition service or any partition within it.

Work area property modes

The information in a work area consists of a set of properties; a property consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether you can modify or remove the property.

Property modes

There are four possible mode values for properties, as shown in the following code example:

Code example: The PropertyModeType definition

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

A property's mode determines three things:
- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The work area service does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. Modifying key-value pairs describes the restrictions each mode places on modifying the value and deleting the property. Changing modes describes the restrictions on changing the mode.

Changing modes

The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:
- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

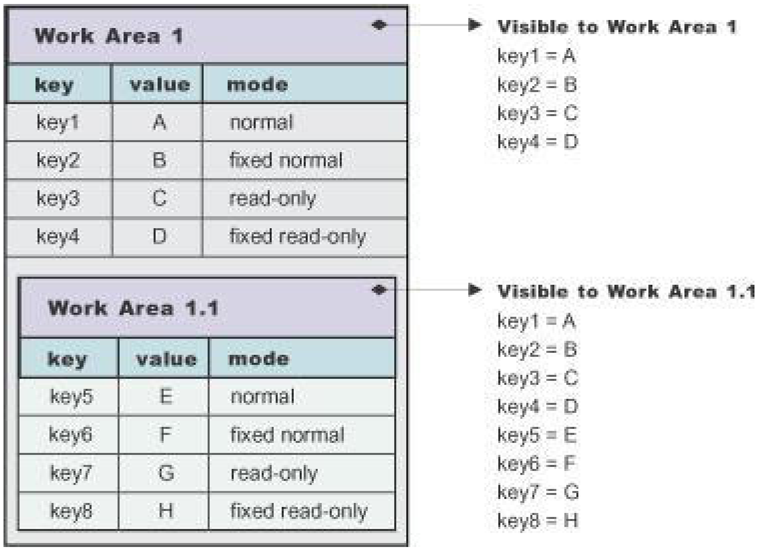
Note: The key, value, and mode of any property can be effectively changed by terminating (completing) the work area in which the property was created and creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas

Applications can nest work areas to define and scope properties for specific tasks without having to make the work areas available to all parts of the application.

When an application creates a work area, a work area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area and becomes the current work area. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In the figure below, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work area are not visible. From the nested work area, the properties in both work areas are visible.



Work Area 1

key	value	mode
key1	A	normal
key2	B	fixed normal
key3	C	read-only
key4	D	fixed read-only

Work Area 1.1

key	value	mode
key5	E	normal
key6	F	fixed normal
key7	G	read-only
key8	H	fixed read-only

Visible to Work Area 1
key1 = A
key2 = B
key3 = C
key4 = D

Visible to Work Area 1.1
key1 = A
key2 = B
key3 = C
key4 = D
key5 = E
key6 = F
key7 = G
key8 = H

Figure 106. Defining new properties in nested work areas

Nesting can also affect the apparent settings of the properties. Properties can be deleted from or directly modified only within the work areas in which they were set, but nested work areas can also be used to temporarily override information in the property without having to modify the property. Depending on the modes associated with the properties in the enclosing work area, the modes and the values of keys in the enclosing work area can be overridden within the nested work area.

The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

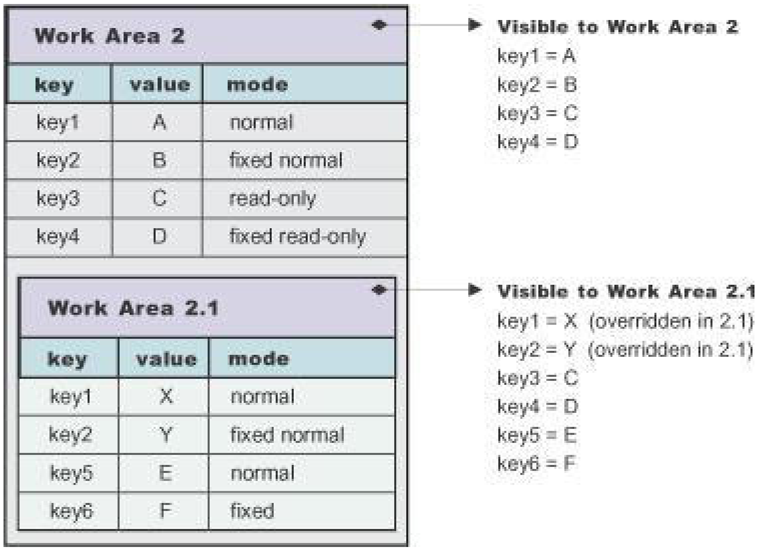
- Modes that permit a nested work area to override the mode or the value of a key locally. The modes that permit overriding are:
  - Normal
  - Fixed normal

- Modes that do not permit a nested work area to override the mode or the value of a key locally. The modes that do not permit overriding are:
  – Read-only
  – Fixed read-only

If an enclosing work area defines a property with one of the modes that can be overridden, a nested work area can specify a new value for the key or a new mode for the property. The new value or mode becomes the value or mode seen by subsequently nested work areas. Changes to the mode are governed by the restrictions described in Changing modes. If an enclosing work area defines a property with one of the modes that cannot be overridden, no nested work area can specify a new value for the key.

A nested work area can delete properties from enclosing work areas, but the changes persist only for the duration of the nested work area. When the nested work area is completed, any properties that were added in the nested area vanish and any properties that were deleted from the nested area are restored.

The following figure illustrates the overriding of properties from an enclosing work area. The nested work area redefines two of the properties set in the enclosing work area. The other two cannot be overridden. The nested work area also defines two new properties. From the outermost work area, the properties set or redefined in the nested work are not visible. From the nested work area, the properties in both work areas are visible, but the values seen for the redefined properties are those set in the nested work area.



Work Area 2

key	value	mode
key1	A	normal
key2	B	fixed normal
key3	C	read-only
key4	D	fixed read-only

Work Area 2.1

key	value	mode
key1	X	normal
key2	Y	fixed normal
key5	E	normal
key6	F	fixed

Visible to Work Area 2
key1 = A
key2 = B
key3 = C
key4 = D

Visible to Work Area 2.1
key1 = X (overridden in 2.1)
key2 = Y (overridden in 2.1)
key3 = C
key4 = D
key5 = E
key6 = F

Figure 107. Redefining existing properties in nested work areas

Distributed work areas

Work area context propagates to a target object on a remote invocation on both bidirectional and non-bidirectional defined work area partitions. The propagation of work area context operates differently depending on whether a work area partition is defined as bidirectional. If the partition is defined as bidirectional, the context propagates from a target object back to the originator.

Non-bidirectional work area partitions (UserWorkArea partition)

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

Bidirectional work area partitions

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, this information is propagated to any remote objects it invokes. In a partition that is not defined as bidirectional, a target application must begin a nested work area before making changes to the imported work area. However, if a partition is defined as bidirectional, a target application need not begin a nested work area before operating on an imported work area. By not beginning a nested work area, any new context set into the work area, or any context changes made by the target application, is not only propagated on future remote invocations but is also propagated back to the originating application (that is, the one who initiated the remote invocation) thus allowing bidirectional propagation of work area context. If the target application does not want new or changed context to propagate back to the originating application, then the target application must begin a nested work area to scope the context to its process. However, the new or changed context in the nested work area propagates on any future remote invocation the target application may make.

WorkArea service: Special considerations

Developers who use work areas should consider the following issues that could potentially cause problems: interoperability between the EJB and CORBA programming models; and the use of work areas with Java's Abstract Windowing Toolkit.

EJB and CORBA interoperability

Although the work area service can be used across the EJB and CORBA programming models, many composed data types cannot be successfully used across those boundaries. For example, if a SimpleSampleCompany instance is passed from the WebSphere environment into a CORBA environment, the CORBA application can retrieve the SimpleSampleCompany object encapsulated within a CORBA Any object from the work area, but it cannot extract the value from it. Likewise, an IDL-defined struct defined within a CORBA application and set into a work area is not readable by an application using the UserWorkArea class.

best-practices: Applications can avoid this incompatibility by directly setting only primitive types, like integers and strings, as values in work areas, or by implementing complex values with structures designed to be compatible, like CORBA valuetypes.

Also, CORBA Anys that contains either the tk_null or tk_void typecode can be set into the work area by using the CORBA interface. However, the work area specification cannot allow the Java 2 Platform, Enterprise Edition (J2EE) implementation to return null on a lookup that retrieves these CORBA-set properties without incorrectly implying that there is no value set for the corresponding key. For example, when a user attempts to retrieve a nonexistent key from a work area, the work area service returns null to indicate that the specified key does not contain a value, implying that the key itself is not in use or does not exist. In the case where CORBA Anys contains either tk_null or tk_void, when a user requests the key associated with one of these values, the work area service returns null as expected. In this case, the key may actually exist and the work area service was simply returning the key's value of null. Therefore, when working with CORBA Anys, a user must not make any implications when a null is returned from a work area because it could mean that either there isn't a property associated with the given key, or that there is a property associated with the given key and it contains a tk_null or tk_void, for example, a null in the J2EE environment. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Using work areas with Java's Abstract Windowing Toolkit (AWT)

Work areas must be used cautiously in applications that use Java's Abstract Windowing Toolkit (AWT). The AWT implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, such as pressing a button, the work area might not be available to any other part of the application after the execution of the event completes.

Chapter 25. Web applications

This page provides a starting point for finding information about web applications, which are comprised of one or more related files that you can manage as a unit, including:

- HTML files
- Servlets can support dynamic web page content, provide database access, serve multiple clients at one time, and filter data.
- Java ServerPages (JSP) files enable the separation of the HTML code from the business logic in web pages.

IBM extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming. Learn about web applications by visiting the following topics:

Learn about web applications

Learn how you can use servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files to create more dynamic and portable web applications.

Web applications

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

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

Web modules

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

A web module contains:

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

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

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

Web container request attributes

A web container provides three custom HttpServletRequest attributes that can be used to provide a servlet or a trust association interceptor (TAI) with the certificate information for a request.

These HttpServletRequest attributes provide information about a client, such as a web server plug-in, that is directly connected to the web container:

- The com.ibm.websphere.ssl.direct_connection_peer_certificates attribute contains a X509Certificate[] object of the certificate for a direct peer.

- The com.ibm.websphere.ssl.direct_connection_cipher_suite attribute contains a string object of a direct cipher suite.
- The com.ibm.websphere.webcontainer.is_direct_connection attribute contains a boolean object that indicates whether the connection was made through a Web server, or was made directly to WebSphere Application Server.

These attributes are different from the usual JEE defined certificate properties which provide information about the end user who is typically connected to the web server. These attributes are available to all applications and can be used when appropriate.

web.xml file

The `web.xml` file provides configuration and deployment information for the web components that comprise a web application.

The Java Servlet specification defines the `web.xml` deployment descriptor file in terms of an XML schema document. For backwards compatibility, any `web.xml` file that is written to Servlet 2.2 or above that worked in previous versions of WebSphere Application Server are supported by the web container.

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

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

Location

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

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

Usage notes

- Is this file read-only?

  No
- Is this file updated by a product component?

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

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

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

Sample file entry

Note: The web.xml file does not represent the entire configuration that is available for the web application. There are other servlets filters, and listeners that can be defined using programmatic configurations, annotations, and web fragments.

Note: Marking the web application metadata complete will prevent annotations and web fragments from being able to configure components.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
 <display-name>Servlet 3.0 application</display-name>
 <filter>
    <filter-name>ServletMappedDoFilter_Filter</filter-name>
    <filter-class>tests.Filter.DoFilter_Filter</filter-class>
    <init-param>
      <param-name>attribute</param-name>
      <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
    </init-param>
  </filter>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>/DoFilterTest</url-pattern>
  <dispatcher>REQUEST</dispatcher>
 </filter-mapping>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>/IncludedServlet</url-pattern>
  <dispatcher>INCLUDE</dispatcher>
 </filter-mapping>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>ForwardedServlet</url-pattern>
  <dispatcher>FORWARD</dispatcher>
 </filter-mapping>
 <listener>
  <listener-class>tests.ContextListener</listener-class>
 </listener>
 <listener>
  <listener-class>tests.ServletRequestListener.RequestListener</listener-class>
 </listener>
 <servlet>
  <servlet-name>welcome</servlet-name>
  <servlet-class>WelcomeServlet</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>ServletErrorPage</servlet-name>
  <servlet-class>tests.Error.ServletErrorPage</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>IncludedServlet</servlet-name>
  <servlet-class>tests.Filter.IncludedServlet</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>ForwardedServlet</servlet-name>
  <servlet-class>tests.Filter.ForwardedServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>welcome</servlet-name>
  <url-pattern>/hello.welcome</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>ServletErrorPage</servlet-name>
  <url-pattern>/ServletErrorPage</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>IncludedServlet</servlet-name>
  <url-pattern>/IncludedServlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>ForwardedServlet</servlet-name>
  <url-pattern>/ForwardedServlet</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>hello.welcome</welcome-file>
 </welcome-file-list>
 <error-page>
  <exception-type>java.lang.ArrayIndexOutOfBoundsException</exception-type>
  <location>/ServletErrorPage</location>
 </error-page>
  <error-page>
   <error-code>404</error-code>
   <location>/error404.html</location>
  </error-page>
</web-app>
```

Note: For each `<error-page>` declaration, select either `<exception-type>` or `<error-code>`, but not both. The `<location>` tag is required.

Default Application

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

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

Snoop servlet

Use the Snoop servlet to retrieve information about a servlet request. This servlet returns the following information:
- Servlet initialization parameters
- Servlet context initialization parameters
- URL invocation request parameters
- Preferred client locale
- Context path
- User principal
- Request headers and their values
- Request parameter names and their values
- HTTPS protocol information
- Servlet request attributes and their values
- HTTP session information
- Session attributes and their values

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

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

HelloHTML servlet

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

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

transition: The PageList Servlet custom extension is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release. Re-architect your legacy applications to use javax.servlet.filter classes instead of com.ibm.servlet classes. Starting from the Servlet 2.3

specification, javax.servlet.filter classes you can intercept requests and examine responses. You can also use javax.servlet.filter classes to achieve chaining functionality, as well as embellishing or truncating responses.

HitCount application

Use the HitCount demonstration application to demonstrate how to increment a counter using a variety of methods, including:

- A servlet instance variable
- An HTTP session
- An enterprise bean

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

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

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

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

JavaServer Pages

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

JSP files support a division of roles:

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

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

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

WebSphere Application Server Version 8.5 supports the JSP 2.1 specification.

Servlets

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

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

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

You can define servlets as welcome files. Non-servlet resources are served only when the fileServingEnabled attribute is set to `true` in the IBM extensions XMI file, ibm-web-ext.xmi, located in each

Web module WEB-INF directory or by using an assembly tool to set the property in the source `.war` file. Serving welcome files is connected to serving static content. Therefore the fileServingEnabled attribute is set in the web module.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Context parameters

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

Servlet mappings

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

Web fragments

When developing web applications, if multiple web modules use the same components, consider including the components in a Web fragment Java archive (JAR) file. The web fragment JAR file contains both the configuration metadata and component class files. This practice enables easier copying from application to application.

Web module deployment descriptor fragments (web fragments) provide the same configuration metadata that a web.xml file provides, but they exist as a web-fragment.xml file that is packaged inside a JAR file in the WEB-INF/lib directory.

Framework developers provide JAR files that are included in a web application which uses that specific framework. If that framework uses servlets, filters, or other web module configuration, web fragments provide the ability to simply drag the JAR file into an application without requiring changes to the existing web module configuration. Previously, web application developers were required to augment their configuration with additional metadata required by the framework. Another use case is the aforementioned need to use the same components across web modules. Also, the use of mock objects or stubs might be made easier with Web fragments.

Scanning for web fragments decreases performance for each JAR file that it checks for a web-fragment.xml file. The higher the number of JAR files in a web application, the higher the performance impact. If performance concerns demand, disable scanning for web fragments by setting metadata-complete to `true` and include any necessary configuration in the web.xml file.

Note: Disabling the scanning of web fragments also disables annotation scanning. Therefore, if you need either of these, both are scanned.

Important: Set the metadata-complete element in the web.xml file to `true` to disable fragment scanning. Use the absolute-ordering tag in the web.xml file to force an order for scanning web fragments or scan a subset of the web fragments. Use the relative-ordering tag in web-fragment.xml files to specify order relative to another fragment.

Including fragments in a web application might inadvertently expose endpoints to security risks if you are unaware of servlets, filters, or security constraints that are included in a web fragment. Verify that all configured servlets, filters, and security constraints are functioning as expected.

Note: If there is a conflict in the web fragments, applications will not deploy. If there is conflict when installing the application, view the SystemOut.log file to understand which items are conflicting.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

General rules for merging web fragments and annotations in the Servlet 3.0 specification:

- JAR files are only scanned for annotations or web fragment metadata if they are in the WEB-INF/lib directory. Shared libraries are not scanned for annotations or web fragment metadata.
- Annotations that are in classes in the WEB-INF/classes directory are merged first and take precedence over any metadata in the WEB-INF/lib directory.
- All JAR files in the WEB-INF/lib directory are considered Web fragments, regardless of whether they contain a web-fragment.xml file. If a web-fragment.xml file does not exist in a JAR file, it is considered to have an implicit, empty web-fragment.xml file.
- Annotations for each JAR file in the WEB-INF/lib directory are merged into the corresponding web-fragment.xml file before checking for conflicting web fragments. Therefore, annotations conflicts in different fragments prevents the application from deploying.
- If you define web fragment ordering, both the annotation and Web fragment metadata for one web fragment is merged before another fragment that is later in the order.
- Because all JAR files in the WEB-INF/lib directory are considered fragments, the `<others>` element in an `<absolute-ordering>` element applies to all JAR files that are not mentioned elsewhere in the ordering.

Asynchronous servlet best practices

The asynchronous servlet feature enables you to process incoming requests and responses without being bound to the original thread that initiated the request.

Consider the following best practices when using asynchronous servlets:

- Applications should not spawn a new thread for each asynchronous operation needed. At a minimum, applications should use a thread pool or use the AsyncContext start(Runnable) method.

- On the client/browser side, you can use AJAX to enable certain portions of the page to be updated asynchronously.
- The servlet container ensures that calls to complete or dispatch do not start until the web container thread that initiated the **startAsync** command exists. However, the servlet container does not handle multiple threads using the same request and response simultaneously. The application can handle its own concurrency or synchronization issues in this case, but it is not recommended because it can be prone to deadlock or race conditions. If the dispatch or complete method is called from a customer-created thread or runnable started with start(Runnable), then dispatch or complete can start immediately on a new thread and any further modifications to the request or response from the thread that initiated these calls is dangerous. Two threads will have access to the request and response, which can have indeterminate results if both threads are modifying those objects. Therefore, do not call any methods on the request or response after a dispatch from the same thread that called the dispatch. Do not call any methods on the request or response after a complete operation is called.
- Asynchronous listeners have an onTimeout method that starts when a time limit is reached for the asynchronous operation. However, the asynchronous operation might still be running on one thread while the onTimeout runs on a different thread. This scenario is the most common way that multiple threads inadvertently use the same request and response simultaneously. A simple approach to this scenario is to use a shared AtomicBoolean method from both the AsyncListener and the asynchronous operation, as follows:

  ```
  AtomicBoolean isOkayToRun = (AtomicBoolean) request.getAttribute("isOkayToRun");
  if (isOkayToExecute.setAndGet(false)){
    //do a dispatch
  }
  ```

  With this approach, only one thread can obtain access to write to the response.
- The web container attempts to cancel any runnables that are queued by calls to the start(Runnable) method when the timeout is reached. However, runnables that have already started cannot be interrupted because the interruption leads to leaking memory.
- The number of threads doing the timeout notifications is very small. Attempting any intensive operation or any write operation from a timeout is not recommended because even a small write operation might take awhile if the client has a slow connection. When you disable the asynchronous timeout, it is easier to run into OutOfMemory errors or to exhaust the number of TCP channel connections. The default timeout is 30 seconds.
- You can configure some asynchronous servlet options, such as timeout settings and the AsyncContext start(Runnable) method, in the administrative console by clicking **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Web Container Settings** > **Web container**. Refer to the Web container settings topic to learn about configuring the Web container.

Important: Asynchronous request dispatcher (ARD) and remote request dispatcher (RRD) are not supported when using asynchronous servlets.

Tip: View the Web application counters topic to learn about metrics for asynchronous servlets.

Web container properties

Learn about system properties, custom properties, and application properties for the web container.

Web container system property

You can change the value of the javax.servlet.context.tempdir servlet context attribute to be relative to a different directory by setting the com.ibm.websphere.servlet.temp.dir system property. This system property affects all applications on a server-wide basis. For example, if you set `com.ibm.websphere.servlet.temp.dir` to `/foo`, the application temp directory is `/foo/node1/server1/fragmentTest/fragmentTest24.war`. If you want to change the value at an application level, use the scratchdir JavaServer Pages (JSP) attribute. View the JSP engine configuration parameters topic for more information about the scratchdir attribute.

Web container custom property

Custom properties are specific to a server. Set a web container custom property in the administrative console. See the Web container custom properties topic for instructions on setting custom properties for web container and to see a list of available web container custom properties.

Web container application property

Application properties are specific to an application. You define application properties in the application deployment descriptor. Enable a Java Enterprise Edition (Java EE) application to configure asynchronous servlet by using a servlet or filter initial parameter. An example of a web.xml definition of a servlet or filter follows:

```
<init-param>
 <param-name>com.ibm.ws.webcontainer.async-supported</param-name>
 <param-value>true</param-value>
</init-param>
```

Web container behavior notes:

Learn about behavior notes for the web container.

Access multipart/form-data

Define a servlet to process multipart/form data by including a @MultipartConfig annotation in the servlet, or by specifying a multipart-config element for the servlet in the application `web.xml` file. An annotation might look like the following example:

```
@MultipartConfig(fileSizeThreshold=1000000, location="temp", maxFileSize=5000000, maxRequestSize=5000000)
```

If a servlet is not defined to process multipart/form data, or the include file is not within the limits that are set by the configuration data, the following behavior is observed in response to a request containing multipart/form data:

```
ServletRequest.getParameter() will return null for a request to obtain a form field.
HttpServletRequest.getPart() or HttpServletRequest.getParts() will throw an
appropriate exception.  The exception message is an indication of the cause of the exception.
```

Java EE application resource declarations

You can configure your Java Enterprise Edition (Java EE) applications to declare dependencies on external resources and configuration parameters. These resources might be injected into the application code, or might be accessed by the application through the Java Naming and Directory Interface (JNDI).

Resource references allow an application to define and use logical names that you can bind to resources when the application is deployed.

The following resource types can be declared by Java EE applications: simple environment entries, Enterprise JavaBeans (EJB) references, web service references, resource manager connection factory references, resource environment references, message destination references, persistence unit references, and persistence context references.

Simple Environment Entries

You can define configuration parameters in your Java EE applications to customize business logic using simple environment entries. As described in the Java EE 6 application, simple environment entry values might be one of the following Java types: String, Character, Bye, Short, Integer, Long, Boolean, Double, Float, Class, and any subclass of Enum.

Note: The Java type, Class, and any subclass of Enum are new in Java EE 6.

The application provider must declare all of the simple environment entries accessed from the application code. The simple environment entries are declared using either annotations (javax.annotation.Resource) in the application code, or using env-entry elements in the XML deployment descriptor.

In the following example from an application, annotations declare environment entries:

```
// Retry interval in milliseconds
@Resource long retryInterval = 3000;
```

In the previous example, the field default value is 3000. You can use an env-entry-value, which you define in the XML deployment descriptor to change this value.

In the following example, an application declares a simple environment entry of type Class, and defines the Class to be injected using an env-entry-value element in the XML deployment descriptor.

```
@Resource(name=TraceFormatter) Class<?> traceFormatter;


<env-entry>
    <env-entry-name>TraceFormatter</env-entry-name>
    <env-entry-value>com.sample.trace.StdOutTraceFormatter</env-entry-value>
</env-entry>
```

In the previous example, the field value is set to the com.sample.trace.StdOutTraceFormatter Class object.

In the following example, an application which declares a simple environment entry called validationMode as a subclass of Enum in the com.sample.Order class, and configures the Enum value of CALLBACK to inject using elements in the XML deployment descriptor.

```
<env-entry>
    <env-entry-name>JPAValidation</env-entry-name>
    <env-entry-type>javax.persistence.ValidationMode</env-entry-type>
    <env-entry-value>CALLBACK</env-entry-value>
    <injection-target>
        <injection-target-class>com.sample.Order</injection-target-class>
        <injection-target-name>validationMode</injection-target-name>
    </injection-target>
</env-entry>
```

In the previous example, the validationMode field is set to the CALLBACK Enum value. Use the same approach when you use annotations and XML code to declare simple environment entries; for example:

```
@Resource (name=JPAValidation)
javax.persistence.ValidationMode validationMode;


<env-entry>
    <env-entry-name>JPAValidation</env-entry-name>
    <env-entry-value>CALLBACK</env-entry-value>
</env-entry>
```

Note: The simple environment entry support of the Java type, Class, and any subclass of Enum is new for Java EE 6. Previously, you might have developed your applications to declare these types as application resources using the resource-env-ref element in the XML deployment descriptor or using the javax.annotation.Resource annotation. For applications that were using these Java types with the javax.annotation.Resource annotation, the com.ibm.websphere.ejbcontainer.EE5Compatibility system property must be enabled. Without the EE5Compatibility system property, the binding-name element of the resource-env-ref element in the `ibm-ejb-jar-bnd.xml` file is ignored, since the data type is now treated as a simple environment entry and not a resource environment reference.

Note: The <lookup-name> deployment descriptor element and the lookup annotation attribute are new in Java EE 6. They specify the JNDI name of a referenced EJB or resource, relative to the java:comp/env naming context. If either is used in a simple environment entry, you cannot use an <env-entry-value> in the same <env-entry>.

Enterprise JavaBeans (EJB) References

As described in the Java EE 6 specification, you can develop your Java EE applications to declare references to enterprise bean homes or enterprise bean instances using logical names called EJB references.

When an application declares a reference to an EJB, the EJB that you reference will be resolved with one of the following techniques.

- Specify an EJB binding in the ibm-ejb-jar-bnd.xml file or ibm-web-bnd.xml file
- Specify an <ejb-link> element in ejb-jar.xml file or web.xml file
- Specify a beanName attribute on the javax.ejb.EJB annotation
- Specify a <lookup-name> element in ejb-jar.xml file or web.xml file
- Specify a lookup attribute on the javax.ejb.EJB annotation
- Locate an enterprise bean that implements the interface declared as the type of the EJB reference (referred to as AutoLink).

The EJB container attempts to resolve the EJB reference using the previous techniques in the order they are listed.

Note: If <lookup-name> or lookup is used in an EJB reference, you cannot use <ejb-link> or beanName in the same EJB reference.

Note: All of the following EJB reference examples assume the SampleCart bean has only a single interface. If the SampleCart bean had multiple interfaces, then add the following suffix to the end of the binding, <ejb-link> element, or beanName attribute : !com.sample.Cart.

In the following example, an application declares an EJB reference using an annotation, and provides a binding for resolution.

```
@EJB(name="Cart")
Cart shoppingCart;


<ejb-ref name="Cart" binding-name="java:app/SampleEJB/SampleCart"/>
```

In the following example, an application declares an EJB reference using an annotation, and provides an ejb-link element for resolution.

```
@EJB(name="Cart")
Cart shoppingCart;


<ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <ejb-link>SampleEJB/SampleCart</ejb-link>
</ejb-local-ref>
```

In the following example, an application declares an EJB reference using an annotation, and provides a lookup attribute for resolution, from the source bean com.sample.SourceBean.

```
@EJB(name="Cart" lookup="java:app/SampleEJB/SampleCart")
Cart shoppingCart;
```

The application could alternatively declare the EJB reference using the <lookup-name> element in the XML deployment descriptor, as in the following example.

```
<ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <lookup-name>java:app/SampleEJB/SampleCart</lookup-name>
    <injection-target>
    <injection-target-class>com.sample.SourceBean</injection-target-class>
    <injection-target-name>ShoppingCart</injection-target-name>
    </injection-target>
</ejb-local-ref>
```

In the following example, an application declares an EJB reference using an annotation, and provides a beanName attribute for resolution.

```
@EJB(name="Cart" beanName="SampleEJB/SampleCart")
Cart shoppingCart;
```

Resource Environment References

As described in the Java EE 6 specification, you can develop applications to declare references to administered objects that are associated with a resource, such as a Connecter CCI InteractionSpec instance, or other object types managed by the EJB container, including javax.transaction.UserTransaction, javax.ejb.EJBContext, javax.ejb.TimerServcie, org.omg.CORBA.ORB, javax.validation.Validator, javax.validation.ValidatorFactory, or javax.enterprise.inject.spi.BeanManager.

When an application declares a reference to an administered object, you must provide a binding to the administered object when the application is deployed. You can provide the binding using the administrative console when you deploy the application, or you can add the binding to the WebSphere binding XML file, `ibm-ejb-jar-bnd.xml` or `ibm-web-bnd.xml`.

In the following example, an application declares a resource environment reference, and provides a binding to the resource:

```
@Resource(name="jms/ResponseQueue")
Queue responseQueue;


<session name="StatelessSampleBean">
    <resource-env-ref name="jms/ResponseQueue" binding-name="Jetstream/jms/ResponseQueue"/>
</session>
```

The application could alternatively declare the resource environment reference using the lookup attribute, and not require a binding, as in the following example:

```
@Resource(name="jms/ResponseQueue", lookup="Jetstream/jms/ResponseQueue")
Queue responseQueue;


<resource-env-ref>
    <resource-env-ref-name>jms/ResponseBean</resource-env-ref-name>
    <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

When an application declares a reference to a container managed object type, a binding is not used. The container provides the correct instance of the referenced object. In the following example, an application declares a resource environment reference to a container-managed object:

```
@Resource
javax.validation.Validator validator;
```

Resource References to Resource References

A new lookup field on the @Resource annotation is added with Java EE 6. You can now declare a resource reference to a resource reference as shown in the following example:

```
@Resource(name="java:global/env/jdbc/ds1ref",
              lookup="java:global/env/jdbc/ds1",
              authenticationType=Resource.AuthenticationType.APPLICATION,
              shareable=false)
    DataSource ds1ref;
@Resource(name="java:global/env/jdbc/ds1refref",
              lookup="java:global/env/jdbc/ds1ref",
              authenticationType=Resource.AuthenticationType.APPLICATION,
              shareable=true)
    DataSource ds1refref;
```

The lookup uses the innermost nesting of references, which in this case is "java:global/env/jdbc/ds1ref".

Web applications: Resources for learning

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

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

Programming model and decisions

- J2EE BluePrints for web applications
- Redbook on the design and implementation of Servlets, JSP files, and enterprise beans

Programming instructions and examples

- WebSphere Studio Application Developer Programming Guide
- Sun's Java^{TM} Tutorial on Servlets and JavaServer Pages
- Web delivered samples in the Samples Gallery

Programming specifications

- Java 2 Software Development Kit (SDK)
- Servlet 2.4 Specification
- JavaServer Pages 2.0 Specification
- Differences between JavaScript and ECMAScript
- ISO 8859 Specifications
- Java 2 Platform, Standard Edition (J2SE)

Asynchronous request dispatcher

Asynchronous request dispatcher

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

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

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

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

Client-side aggregation of the asynchronous include is also supported. Web 2.0 programmers often use Asynchronous JavaScript and XML (Ajax) in the Web browser of the client to dynamically retrieve and aggregate remote resources. Unfortunately, this puts the burden on the programmer to aggregate the contents and learn new technologies. Client-side aggregation automatically adds the necessary JavaScript to dynamically update the page. For non-JavaScript clients, you can switch ARD to server-side aggregation, which gives equivalent results. You can deny non-JavaScript clients when using client-side aggregation.

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

Asynchronous request dispatcher application design considerations:

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

Asynchronous request dispatcher client-side implementation

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

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

  are placed in the browser object model since comment nodes have no effect on the page layout.
- Whenever a complete fragment exists, a response can be sent to the client and the comment node with the same ID is replaced. Requests are made until all the fragments are retrieved.

- Verify applications on all supported browsers when using client-side aggregation. Object oriented JavaScript principles are used so that applications only need avoid using the method name getDynamicDataIBMARD. Any previously specified window.onload is started before the ARD onload method.

Asynchronous request dispatcher channel results service

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

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

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

Custom client-side aggregation

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

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

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

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

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

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

Server-side aggregation

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

Asynchronous beans

An AsynchBeans work manager is used to start the includes. If the number of currently requested includes is greater than the work manager maximum thread pool size and this size is not growable, it starts the work on the current thread and skips the placeholder write. Utilizing AsynchBeans suppports propagation of the J2EE context of the original thread including work area, internationalization, application profile, z/OS operation system work load management, security, transaction, and connection context.

Timer

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

Remote request dispatcher

Optionally, ARD can be used in concert with the remote request dispatcher. The remote request dispatcher was introduced in WebSphere Application Server, Network Deployment 6.1. The remote request dispatcher runs the include on a different application server in a core group by serializing the request context into a SOAP message and using web services to call the remote server. This is useful when the expense of creating and sending a SOAP message through web services is outweighed by issuing the request locally. For more information, see the IBM WebSphere Developer Technical Journal: Include remote files in your web application seamlessly with the new Remote Request Dispatcher developerWorks article.

Exceptions

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

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

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

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

Include start
The work manager provides a timeout for how long to wait for an include to start. Since this typically happens immediately, there is not a programmatic way to enable this. However, this is

configurable in the work manager settings. By default, you will not encounter this because of the maximum thread check before scheduling the work. Work can be retried if setRetriable(true) is called on the in use AsyncRequestDispatcherConfig.

Include finish

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

Expiration of results

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

Includes versus fragments

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

Web container

ServletContext

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

ServletRequest

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

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

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

ServletResponse

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

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

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

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

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

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

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

Transactions

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

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

Connection management

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

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

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

Performance

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

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

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

Security

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

Sessions

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

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

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

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

Session management support

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

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

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

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

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

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

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

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

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

Note: Session management configuration is a post-deployment configuration and is tied to existing targets. If you change the target mapping after you configure session management, you must return to the session management configuration page in the administrative console or use wsadmin scripting and apply the changes. Apply the changes to module targets if session management is configured for a web module. Apply the changes to all targets if session management is configured for an application level.

Distributed sessions

In a distributed environment, you can save sessions in a database using database session persistence.

WebSphere Application Server provides the following session mechanisms in a distributed environment:

- **Database session persistence**, where sessions are stored in the database specified.

When a session contains attributes that implement HttpSessionActivationListener, notification occurs anytime the session is activated (that is, session is read to the memory cache) or passivated (that is, session leaves the memory cache). Passivation can occur because of a server shutdown or when the session memory cache is full and an older session is removed from the memory cache to make room for a newer session. It is not guaranteed that a session is passivated in one application server prior to activation in another application.

Scheduled invalidation

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

When used with distributed sessions, this feature has the following benefits:
- You can schedule the scan for invalidated sessions for times of low application server activity, avoiding contention between invalidation scans of database or another WebSphere Application Server instance and read and write operations to service HTTP session requests.
- Significantly fewer external write operations can occur when running with the End of Service Method Write mode because the last access time of the session does not need to be written out on each HTTP request. (Manual Update options and Time Based Write options already minimize the writing of the last access time.)
- The session manager invalidates sessions only at the scheduled time, therefore sessions are available to an application if they are requested prior to the session is invalidated. The following are usage considerations:
- With scheduled invalidation configured, HttpSession timeouts are not strictly enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- HttpSessionBindingListener processing is handled at the configured invalidation times unless the HttpSession.invalidate method is explicitly called.
- The HttpSession.invalidate method immediately invalidates the session from both the session cache and the external store.
- The periodic invalidation thread still runs with scheduled invalidation. If the current hour of the day does not match one of the configured hours, sessions that have exceeded the invalidation interval are removed from cache, but not from the external store. Another request for that session results in returning that session back into the cache.
- When the periodic invalidation thread runs during one of the configured hours, all sessions that have exceeded the invalidation interval are invalidated by removal from both the cache and the external store.
- The periodic invalidation thread can run more than once during an hour and does not necessarily run exactly at the top of the hour.
- If you specify the interval for the periodic invalidation thread using the HttpSessionReaperPollInterval custom property, do not specify a value of more than 3600 seconds (1 hour) to ensure that invalidation processing happens at least once during each hour.

Base in-memory session pool size

The base in-memory session pool size number depends on the session support configuration.
- With in-memory sessions, session access is optimized for up to this number of sessions.
- With distributed sessions, when sessions are stored in a database or in another WebSphere Application Server instance,; the pool size also specifies the cache size and the number of last access time updates saved in manual update mode.

For distributed sessions, when the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

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

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

Overflow in non-distributed sessions

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

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

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

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

HTTP session invalidation

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

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

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

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

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

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

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

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

Write operations

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

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

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

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

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

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

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

HTTP sessions: Resources for learning

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

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

View links to additional information about:

Programming model and decisions

- Improving session persistence performance with DB2

Programming instructions and examples

- Java Servlet documentation, tutorials, and examples site

Programming specifications

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

Asynchronous request dispatcher

Asynchronous request dispatcher

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

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

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

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

Client-side aggregation of the asynchronous include is also supported. Web 2.0 programmers often use Asynchronous JavaScript and XML (Ajax) in the Web browser of the client to dynamically retrieve and aggregate remote resources. Unfortunately, this puts the burden on the programmer to aggregate the contents and learn new technologies. Client-side aggregation automatically adds the necessary JavaScript to dynamically update the page. For non-JavaScript clients, you can switch ARD to server-side aggregation, which gives equivalent results. You can deny non-JavaScript clients when using client-side aggregation.

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

Asynchronous request dispatcher application design considerations

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

Asynchronous request dispatcher client-side implementation

- JavaScript is dynamically written to the response output.
- This JavaScript results in Ajax requests back to a server-side results provider.

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

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

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

Asynchronous request dispatcher channel results service

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

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

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

Custom client-side aggregation

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

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

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

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

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

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

Server-side aggregation

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

Asynchronous beans

An AsynchBeans work manager is used to start the includes. If the number of currently requested includes is greater than the work manager maximum thread pool size and this size is not growable, it starts the work on the current thread and skips the placeholder write. Utilizing AsynchBeans suppports propagation of the J2EE context of the original thread including work area, internationalization, application profile, z/OS operation system work load management, security, transaction, and connection context.

Timer

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

Remote request dispatcher

Optionally, ARD can be used in concert with the remote request dispatcher. The remote request dispatcher was introduced in WebSphere Application Server, Network Deployment 6.1. The remote request dispatcher runs the include on a different application server in a core group by serializing the request context into a SOAP message and using web services to call the remote server. This is useful when the expense of creating and sending a SOAP message through web services is outweighed by issuing the request locally. For more information, see the IBM WebSphere Developer Technical Journal: Include remote files in your web application seamlessly with the new Remote Request Dispatcher developerWorks article.

Exceptions

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

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

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

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

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

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

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

Includes versus fragments

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

Web container

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

ServletRequest
You must clone the request for each include. Otherwise, conflicts between threads might occur. Because applications can wrap the default request objects, your wrappers must implement the

com.ibm.wsspi.webcontainer.servlet.IServletRequest interface, which has one method, the public Object clone method, which creates the CloneNotSupportedException.

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

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

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

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

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

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

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

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

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

Transactions

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

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

Connection management

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

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

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

Performance

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

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

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

Security

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

Chapter 26. Web services

This page provides a starting point for finding information about web services.

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Overview: Online garden retailer web services scenarios

This set of scenarios is inspired by an online retailer called Plants by WebSphere. Plants by WebSphere uses web services support in WebSphere Application Server to improve communications with its suppliers. The more advanced scenarios describe web services support available only in particular editions of the application server. Consult the product documentation to confirm what is supported by your edition.

You might recognize Plants by WebSphere as a sample application available in the Samples section of the Information Center. These scenarios are loosely related. They describe how the fictional online retailer could use a variety of web services technologies, some of which are beyond those currently demonstrated by the sample.

Web services are middleware. Using web services you can connect applications together, no matter how each application is implemented or where it is located. For example, web services can connect retailers to wholesale suppliers. Middleware is not new. What is new in Web services is that this connectivity is based upon open standards and web technologies. Web services operate at a level of abstraction that is similar to the Internet, and they can work with any operating system, hardware platform, or programming language that can be Web-enabled.

The Plants by WebSphere storefront sells plants and gardening supplies. As customers order merchandise, the site checks the merchandise availability in its inventory database. The scenarios show how the inventory system can grow in stages, using various web services technologies to improve its capabilities.

- Before web services

  The Plants by WebSphere application already has web services capabilities. See the following for a description of how the online garden retailer might have operated prior to adopting web services technology. Key web services components are introduced. To determine which components are available your particular editions of WebSphere Application Server, consult the documentation for each edition.
- Static inquiry on supplier

  In this scenario, the garden retailer turns the existing web application into a web service for checking the inventory of its main wholesale garden supplier.
- Dynamic inquiry on supplier

  In this scenario, the garden retailer uses web services to perform an inventory search of several wholesale suppliers.
- Cross supplier inquiry

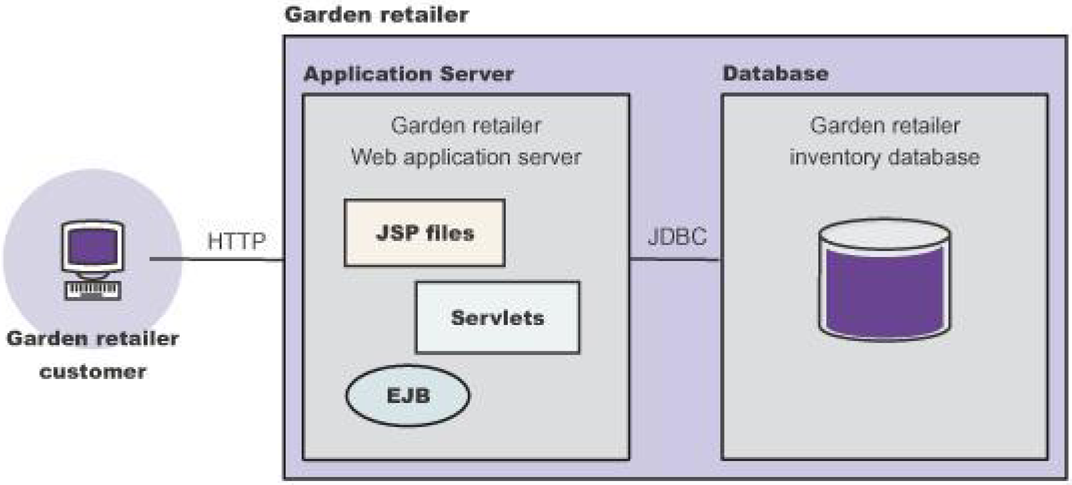
  In this scenario, the garden retailer makes its web service available for use by others who need the service.

At present, these scenarios provide descriptions rather than step by step instructions. To gain experience with web services coding, see the sample application. It provides detailed instructions for building, configuring, and running the Plants by WebSphere sample application and others.

Before web services

Suppose that the Plants by WebSphere storefront does not use web services. The garden retailer has established an impressive Internet storefront enabling customers to shop and order merchandise. To determine whether a customer order can be filled, web applications rely on enterprise beans to query the Plants by WebSphere inventory database. If the item is in stock, the site confirms the order to the customer.

If a customer orders an item that is out of stock, the site notifies the customer that the item is out of stock, and encourages the customer to place the item on backorder. Later, long after the customer has left the Plants by WebSphere site, the site administrator or inventory manager might call or fax the supplier to obtain more inventory.



Introducing web services

Using web services provides Plants by WebSphere an automated way to have out of stock items shipped to its warehouse or directly to customers. If suppliers can be contacted quickly enough, Plants by WebSphere does not have to inform its customers that the item was out of stock. Plants by WebSphere can begin to reduce its own inventory if doing so is a desirable business move.

Web services is built on the following core technologies:

- **XML**

  Extensible Markup Language (XML) solves the problem of data independence. You use XML to describe data and to map that data into and out of any application or programming language.

  To have their applications exchange information such as merchandise price and availability, Plants by WebSphere and its supplier place the data in a set of XML tags to which both parties agree.
- **WSDL (Web Services Description Language)**

  You use this XML-based language to create a description of an underlying application. This Web Services Description Language (WSDL) document contains the description of your application and it is this description that turns an application into a web service, by acting as the interface between the underlying application and other Web-enabled applications.

  Plants by WebSphere has an application capable of querying the supplier inventory. To enable communication with the supplier over the Internet, the company turns the application into a web service.
- **SOAP**

SOAP is the core communications protocol for the Web, and most web services use this protocol to talk to each other.

SOAP is an XML format for web services requests. According to the SOAP specification, SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML-based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.

Because they are external to the Plants by WebSphere intranet, communications with its suppliers utilize SOAP over HTTP. Web services operating within the company intranet can use other transports, such as local Java bindings. The Web Services Invocation Framework (WSIF) component described later in this section can help Plants by WebSphere applications dynamically choose the optimal transport mechanism for a given situation.

- **Web Services for Java Platform, Enterprise Edition (Java EE)**

  The Web Services for Java Platform, Enterprise Edition (Java EE) specification, also known as JSR-109, defines how Java EE applications create and access web services.

  Read about implementing web services applications to learn how to implement a web service interface to an existing application, and deploy your web service within the application server.
- **Java API for XML Web Services (JAX-WS)**

  The JAX-WS programming model simplifies application development through support of a standard, annotation-based model to develop web services applications and clients. The JAX-WS programming model is the successor to the JAX-RPC 1.1 programming model.

  The application server supports both the JAX-WS and JAX-RPC programming models.
- **Java Architecture for XML Binding (JAXB)**

  JAXB is a Java technology that provides and easy way to map Java classes and XML schema in the development of web services applications. JAXB leverages platform-neutral XML data to bind XML schema to Java applications without requiring extensive knowledge of XML programming.
- **Java API for XML-based remote procedure call (JAX-RPC)**

  JAX-RPC, also known as JSR-101, defines how Java applications access web services.

The application server supports web services based on the JAX-WS and JAX-RPC programming models. JAX-WS is a new programming model that simplifies application development through support of a standard, annotation-based model to develop web services applications and clients. A JAX-RPC client and JAX-WS client can be used in the same module; therefore, the online retailer is still able to use its JAX-RPC applications. The application server makes it easy to configure and reuse configurations, so you can seamlessly incorporate new Web services profiles. The JAX-WS standards support interoperable and reliable web services applications. The online retailer can send messages asynchronously, which means that the messages can communicate reliably even if one of the parties is temporarily offline, busy, or not available. By using these new technologies, the online retailer can be confident that its communication is reliable and reaches its destination while interoperating with other vendors.

Refer to the Samples section of the Information Center for additional Samples that demonstrate JAX-WS and JAX-RPC web services.

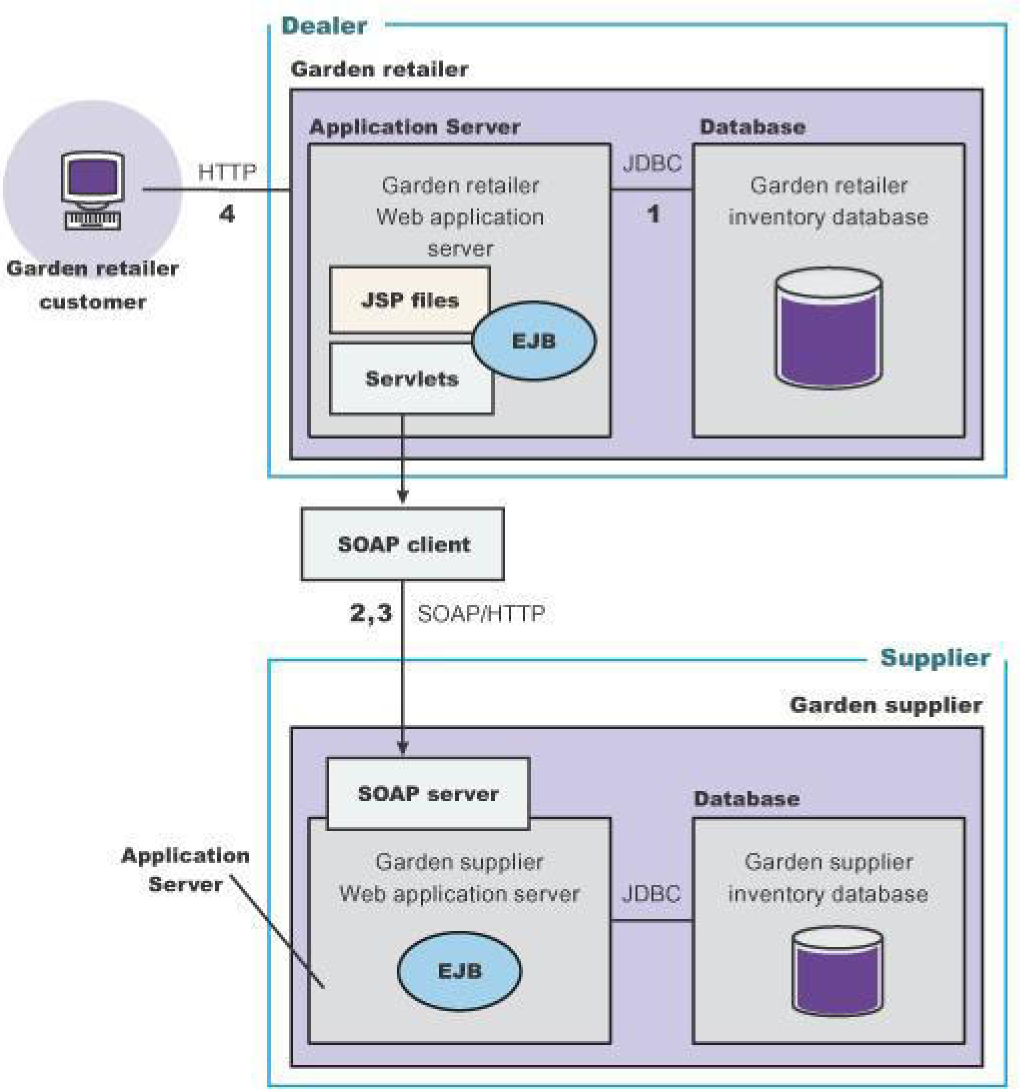
WebSphere software provides additional specifications and standards to help you get the most out of your web services.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Web services online garden retailer scenario: Static inquiry on supplier

In this scenario, an online supply retailer turns an application into a web service for checking the inventory of its main supplier.

Plants by WebSphere is an Internet storefront that sells plants and gardening supplies. The company realizes that its existing web application can be "wrapped" with web services programming interfaces. For example, the BackOrderStock session bean can be turned into a SOAP web services client that interacts with a Supplier Web services application located at the supplier. Specifically, the BackOrderStock session bean functionality is turned into a Web Services for Java Platform, Enterprise Edition (Java EE) client and a Java API for XML-based remote procedure call (JAX-RPC) client.



The application server supports web services based on the JAX-WS and JAX-RPC programming models. JAX-WS is a new programming model that simplifies application development through support of a standard, annotation-based model to develop web services applications and clients. A JAX-RPC client and JAX-WS client can be used in the same module, therefore the online retailer is still able to use its JAX-RPC applications. The application server makes it easy to configure and reuse configurations, so you can seamlessly incorporate new web services profiles. The JAX-WS standards support interoperable and reliable web services applications. The online retailer can send messages asynchronously, which means that the messages can communicate reliably even if one of the parties is temporarily offline, busy, or not available. By using these new technologies, the online retailer can be confident that its communication is reliable and reaches its destination while interoperating with other vendors.

How out of stock items are handled

The following events happen when a customer on the Plants by WebSphere site orders an item that is not available according to the Plants by WebSphere inventory:

1. Plants by WebSphere checks its own inventory.

   The application powering the website checks the Plants by WebSphere inventory database. It discovers that the item is not in stock.
2. Plants by WebSphere uses a web service to check the supplier inventory.

   The application invokes a SOAP client that communicates with a SOAP server at the supplier site to ascertain whether the supplier has the item in stock. The supplier data is sent to Plants by WebSphere.
3. Plants by WebSphere either obtains the out of stock item, or does not.

   If the supplier indicates that the item is in stock, the application powering Plants by WebSphere determines whether to order the item on behalf of the customer. The exchange of data can include checking a price threshold beyond which Plants by WebSphere will not order the wholesale item. It could also include decision-making information such as how long the supplier requires to deliver the item, or a date that the manufacturer plans to discontinue the item.
4. Plants by WebSphere notifies its customer of the outcome, as soon as possible.

   If the supplier can be consulted quickly enough, Plants by WebSphere does not have to bother its customer with concerns about availability. It simply confirms that the item is available, as though the item is in stock at Plants by WebSphere. If the supplier inventory temporary lacks the item, or Plants by WebSphere opted not to order the item from the supplier, Plants by WebSphere can issue an appropriate response to the customer.

Web services technologies used in this scenario

This scenario uses the following web services technologies.

XML (Extensible Markup Language)
XML is used to standardize the exchange of data between Plants by WebSphere and its supplier.

Web Services for Java Platform, Enterprise Edition (Java EE)
Web Services for Java EE, also known as JSR-109, defines how Java EE applications create and access web services.

JAX-RPC
JAX-RPC, also known as JSR-101, defines how Java applications access web services.

JAX-WS
The JAX-WS programming model simplifies application development through support of a standard, annotation-based model to develop Web services applications and clients. The JAX-WS programming model is the successor to the JAX-RPC 1.1 programming model. The application server supports both the JAX-WS and JAX-RPC programming models.

WSDL (Web Services Description Language)
WSDL is used to turn the existing application into a web service, by acting as the interface between the underlying application and other Web-enabled applications.

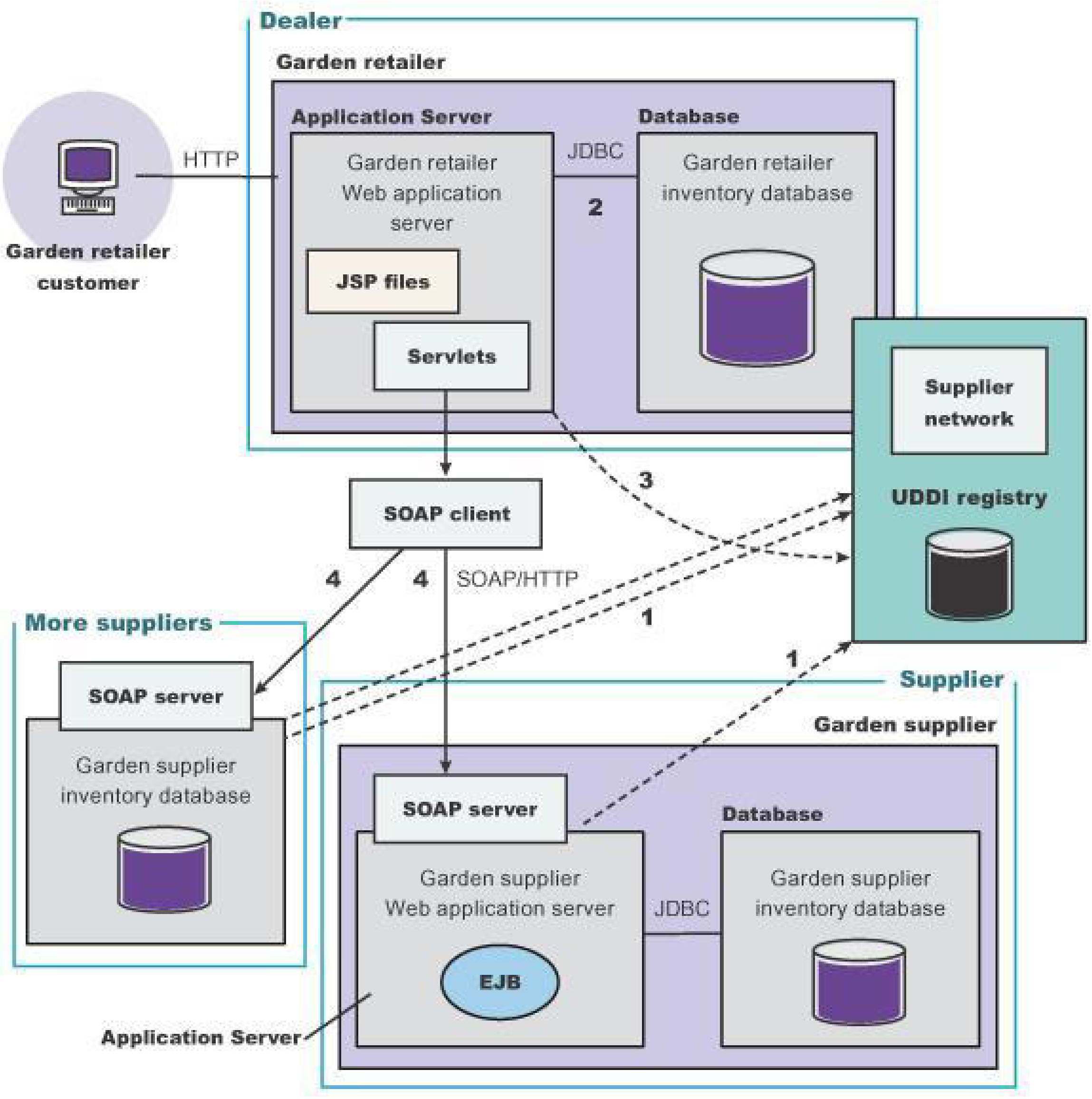
SOAP
SOAP is the protocol by which the web service communicates with the supplier over the Internet.

Web services online garden retailer scenario: Dynamic inquiry on supplier

This document describes a scenario in which an online garden supply retailer uses web services to perform an inventory search of several wholesale suppliers.

In the Plants by WebSphere web services static inquiry on suppliers scenario, the Plants by WebSphere IT staff establishes connections with each supplier separately, and makes changes as suppliers come and go. It would be convenient to query multiple suppliers at the same time, without providing a list of particular suppliers to query. Furthermore, Plants by WebSphere managers would like to be able to shop around quickly according to criteria, such as the lowest wholesale price or fastest availability.

In this scenario, several plant and garden suppliers have published web services to a Universal Description, Discovery, and Integration. (UDDI) registry. Suppliers create inventory web services that use a standard interface. They publish their web services to the centralized registry. Perhaps the registry has been established by the Plant Wholesalers Association. Or maybe a small Internet company established the buyer-seller site after finding that suppliers and retailers each would pay a small monthly rate for the convenience of the service.



Plants by WebSphere also supports JavaBeans endpoints within the web container and enterprise beans endpoints by taking advantage of the Java API for XML Web Services (JAX-WS) programming model support. Using the JAX-WS programming model makes it easy to configure and reuse configurations, so you can seamlessly incorporate new web services profiles. The JAX-WS standards support interoperable and reliable web services applications. The online retailer can send messages asynchronously, which means that the messages can communicate reliably even if one of the parties is temporarily offline, busy, or not available. By using these new technologies, the online retailer can be confident that its communication is reliable and reaches its destination while interoperating with other vendors.

How out of stock items are handled

The following events happen when a customer on the Plants by WebSphere site orders an item that is not available according to the Plants by WebSphere inventory.

1. In advance, the suppliers publish their web services to a UDDI registry for such an occasion.

   In this way, they notify inquiring retailers, such as Plants by WebSphere, that their inventories are available to check.
2. Plants by WebSphere checks its own inventory.

   The application powering the website checks the Plants by WebSphere inventory database. It discovers that the item is not in stock.
3. Plants by WebSphere uses a UDDI4J client to consult the UDDI registry for suppliers whose inventories it can check.

   Plants by WebSphere can invoke a web service that queries the UDDI registry for suppliers and the web service at the site of each supplier is invoked. The administrator is presented with a list of suppliers from which the requested item is available, including the price and availability data.
4. Plants by WebSphere uses the web services to check the supplier inventories.

   The application invokes a Web Services for Java Platform, Enterprise Edition (Java EE) client or a JAX-RPC client that communicates with a SOAP server at the supplier site to ascertain whether the supplier has the item in stock. The supplier data is sent to Plants by WebSphere.

   The application invokes a JAX-RPC client, or a JAX-WS application client can be invoked, that communicates with a SOAP server at the supplier site to ascertain whether the supplier has the item in stock. The supplier data is sent to Plants by WebSphere.
5. Plants by WebSphere either obtains the out of stock item, or does not.

   If the supplier indicates that the item is in stock, the application powering Plants by WebSphere determines whether to order the item on behalf of the customer. The exchange of data can include checking a price threshold beyond which Plants by WebSphere does not order the wholesale item. It could also include decision-making information such as how long the supplier requires to deliver the item, or a date that the manufacturer plans to discontinue the item.
6. Plants by WebSphere notifies its customer of the outcome, as soon as possible.

   If the supplier can be consulted quickly enough, Plants by WebSphere does not have to bother its customer with concerns about availability. It confirms that the item is available, as though the item is in stock at Plants by WebSphere. If the supplier inventory temporary lacks the item, or Plants by WebSphere opted not to order the item from the supplier, Plants by WebSphere issues an appropriate response to the customer.

Web services technologies used in this scenario

This scenario uses the following web services technologies.

XML (Extensible Markup Language)

XML is used to standardize the exchange of data between Plants by WebSphere and its supplier.

Web Services for Java Platform, Enterprise Edition (Java EE)

Web Services for Java Platform, Enterprise Edition (Java EE), also known as JSR-109, defines how Java EE applications create and access web services.

Java API for XML-based remote procedure call (JAX-RPC)
JAX-RPC, also known as JSR-101, defines how Java applications access web services.

JAX-WS
The JAX-WS programming model simplifies application development through support of a standard, annotation-based model to develop Web services applications and clients. The JAX-WS programming model is the successor to the JAX-RPC 1.1 programming model. The application server supports both the JAX-WS and JAX-RPC programming models.

WSDL (Web Services Description Language)
WSDL is used to turn the existing application into a web service, by acting as the interface between the underlying application and other Web-enabled applications.

SOAP SOAP is the protocol by which the web service communicates with the supplier over the Internet.

UDDI registry

By publishing their web services to UDDI, suppliers make them available for Plants by WebSphere and other retailers to discover and reuse. This saves development time, effort and cost, and helps minimize the need to maintain several different implementations of the same application at Plants by WebSphere and various other retailers who need to contact the suppliers for inventory data.

Particular editions of WebSphere Application Server provide a private UDDI registry that can be used in an intranet environment.

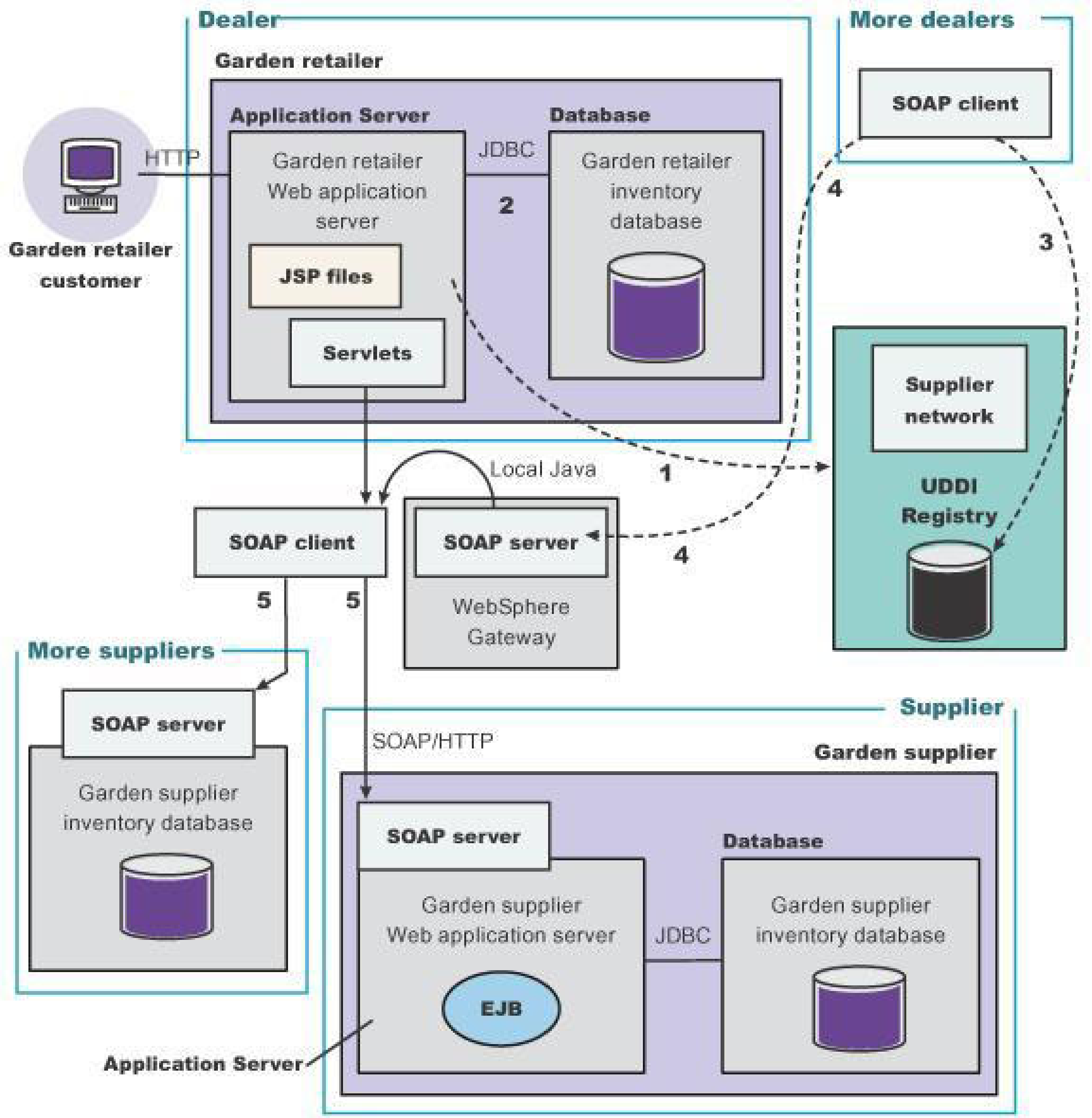
Web services online garden retailer scenario: Cross supplier inquiry

This document describes a scenario in which an online garden supply retailer uses web services to integrate its inventory system with the inventory systems of other retailers. Also using web services, the main Internet storefront can check supplier inventories on behalf of itself or other retailers.

The marketers at Plants by WebSphere confirm with market data that people are likely to purchase plants and gardening supplies in tandem with purchases of other goods, such as gardening books. To increase the visibility of Plants by WebSphere, the company arranges with various other merchant sites to include Plants by WebSphere inventory as part of their own.

At one site, web services and other technologies are used to insert data about Plants by WebSphere items into web pages that match the look and feel of the rest of the site. When a customer orders a Plants by WebSphere item at a site other than Plants by WebSphere, the second site relies on the Plants by WebSphere inventory web service to make sure that the item is in stock, and to query suppliers as needed.

The second site does not have to implement its own web services to perform the same function as those developed by Plants by WebSphere. The second site might want to implement sophisticated function by creating its own web service.



Plants by WebSphere also supports JavaBeans endpoints within the web container and enterprise beans endpoints by taking advantage of the Java API for XML Web Services (JAX-WS) programming model support. Using the JAX-WS programming model makes it easy to configure and reuse configurations, so you can seamlessly incorporate new web services profiles. And, the new standards support interoperable and reliable web services applications. The online retailer can send messages asynchronously, which means that the messages can communicate reliably even if one of the parties is temporarily offline, busy, or not available. By using these new technologies, the online retailer can be confident that its communication is reliable and reaches its destination while interoperating with other vendors.

How out of stock items are handled

The following events happen when a customer orders an item from one of the sites that re-sells items from Plants by WebSphere.

1. In advance, Plants by WebSphere publishes its Web service to a public Universal Description, Discovery and Integration (UDDI) registry.

By publishing the web service, other retailers are made aware of the inventory web service available from Plants by WebSphere. In this scenario, Plants by WebSphere enables the web service to check its own inventory, as well as that of suppliers.

2. The re-seller checks the Plants by WebSphere inventory.

   The application powering the website checks the Plants by WebSphere inventory database. It discovers that the item is not in stock.
3. The re-seller consults the UDDI registry for suppliers whose inventories it can check.
4. The re-seller uses the web services to check the supplier inventories.

   The application invokes a Java API for XML-based remote procedure call (JAX-RPC) SOAP client, or a JAX-WS SOAP client that communicates with a SOAP server at the supplier site to ascertain whether the supplier has the item in stock. The supplier data is sent to the reseller.
5. The re-seller either obtains the out of stock item, or does not.
6. The re-seller notifies its customer of the outcome, as soon as possible.

Web services technologies used in this scenario

This scenario uses the following web services technologies.

XML (Extensible Markup Language)
XML is used to standardize the exchange of data between Plants by WebSphere and its supplier.

WSDL (Web Services Description Language)
WSDL is used to turn the existing application into a web service, by acting as the interface between the underlying application and other Web-enabled applications.

SOAP SOAP is the protocol by which the web service communicates with the supplier over the Internet.

Service-oriented architecture

A *service-oriented architecture* (SOA) is a collection of services that communicate with each other, for example, passing data from one service to another or coordinating an activity between one or more services.

Companies want to integrate existing systems to implement Information Technology (IT) support for business processes that cover the entire business value chain. A variety of designs are used, ranging from rigid point-to-point electronic data interchange (EDI) to web auctions. By using the Internet, companies can make their IT systems available to internal departments or external customers, but the interactions are not flexible and are without standardized architecture.

Because of this increasing demand for technologies that support connecting and sharing resources and data, a need exists for a flexible, standardized architecture. SOA is a flexible architecture that unifies business processes by structuring large applications into building blocks, or small modular functional units or services, for different groups of people to use inside and outside the company. The building blocks can be one of three roles: service provider, service broker, or service requestor. See Web services approach to a service-oriented architecture to learn more about these roles.

Requirements for an SOA:

To efficiently use an SOA, follow these requirements:

- **Interoperability between different systems and programming languages**.

  The most important basis for a simple integration between applications on different platforms is to provide a communication protocol. This protocol is available for most systems and programming languages.
- **Clear and unambiguous description language**.

To use a service offered by a provider, it is not only necessary to be able to access the provider system, but the syntax of the service interface must also be clearly defined in a platform-independent fashion.

- **Retrieval of the service**.

  To support a convenient integration at design time or even system run time, a search mechanism is required to retrieve suitable services. Classify these services as *computer-accessible*, *hierarchical* or *taxonomies* based on what the services in each category do and how they can be invoked.

Web services approach to a service-oriented architecture

You can use web services in a service-oriented architecture (SOA) environment.

You can use web services to implement a SOA. A major focus of web services is to make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. A service can rely on another service to achieve its goals.

Each SOA building block can assume one or more of three roles:

- **Service provider**

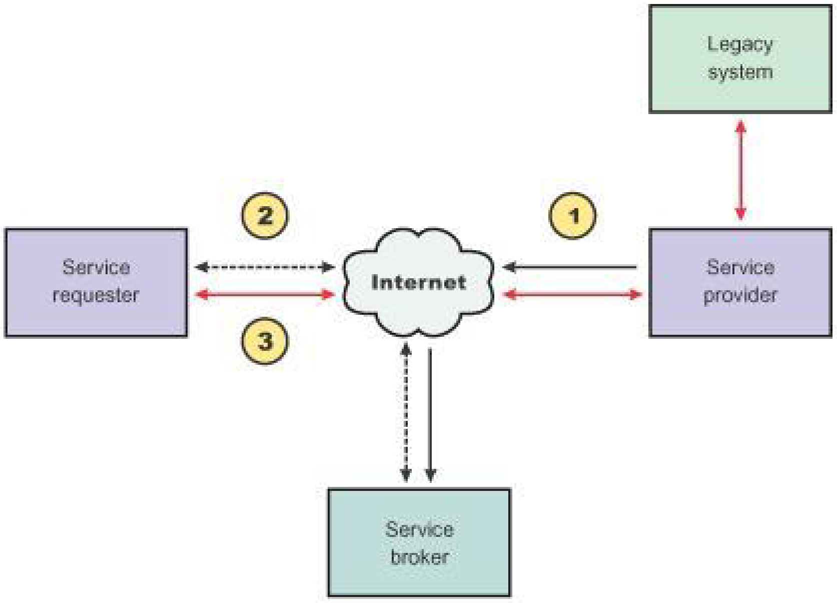
  The service provider creates a web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or how to exploit free services for other value. The provider also has to decide which category to list the service in for a given broker service and what sort of trading partner agreements are required to use the service.

- **Service broker**

  The service broker, also known as *service registry*, is responsible for making the web service interface and implementation access information available to any potential service requestor. The implementer of the broker decides the scope of the broker. Public brokers are available through the Internet, while private brokers are only accessible to a limited audience, for example, users of a company intranet. Furthermore, some decisions need to be made about the amount of the offered information. Some brokers specialize in many listings. Others offer high levels of trust in the listed services. Some cover a broad landscape of services and others focus within an industry. Some brokers catalog other brokers. Depending on the business model, brokers can attempt to maximize look-up requests, the number of listings or the accuracy of the listings. The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about web services.

- **Service requester**

  The service requestor or web service client locates entries in the broker registry using various find operations and then binds to the service provider to invoke one of its web services.



Characteristics of the SOA:

The presented SOA illustrates a loose coupling between the participants, which provides greater flexibility in the following ways:

- A client is coupled to a service. Therefore, the integration of the server takes place outside the scope of the client application programs.
- Old and new functional blocks or applications and systems, are encapsulated into components that work as services.
- Functional components and their interfaces are separate so that new interfaces can be plugged in more easily.
- Within complex applications, the control of business processes can be isolated. A business rule engine can be incorporated to control the workflow of a defined business process. Depending on the state of the workflow, the engine calls the respective services.
- Services can be incorporated dynamically during run time.
- Bindings are specified using configuration files and can be easily adapted to new needs.

Properties of a service-oriented architecture:

The service-oriented architecture offers the following properties:

- **Web services are self-contained**

  On the client side, no additional software is required. A programming language with Extensible Markup Language (XML) and HTTP client support is enough to get you started. On the server side, a web server and a SOAP server are required. It is possible to enable an existing application for web services without writing a single line of code.
- **Web services are self-describing**

  Neither the client nor the server knows or cares about anything besides the format and content of the request and response messages (loosely coupled application integration). The definition of the message format travels with the message; no external metadata repositories or code generation tool are required.
- **Web services can be published, located, and invoked across the Internet**

  This technology uses established lightweight Internet standards such as HTTP and it leverages the existing infrastructure. Some other standards that are required include, SOAP, Web Services Description Language (WSDL), and UDDI.

- **Web services are language-independent and interoperable**

  The client and server can be implemented in different environments. Existing code does not have to change in order to be web services-enabled.
- **Web services are inherently open and standard-based**

  XML and HTTP are the major technical foundation for web services. A large part of the Web service technology has been built using open-source projects.
- **Web services are dynamic**

  Dynamic e-business can become reality using web services because with UDDI and WSDL you can automate the web service description and discovery.
- **Web services are composable**

  Simple web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer web services from a web service implementation. Web services can be chained together to perform higher-level business functions. This chaining shortens development time and enables best-of-breed implementations.
- **Web services are loosely coupled**

  Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that supports a more flexible reconfiguration for an integration of the services.
- **Web services provide programmatic access**

  The approach provides no graphical user interface; it operates at the code level. Service consumers need to know the interfaces to web services, but do not need to know the implementation details of services.
- **Web services provide the ability to wrap existing applications**

  Already existing stand-alone applications can easily integrate into the SOA by implementing a web service as an interface.

Web services business models supported in SOA

This article explains the concept and business models that can be implemented by using web services in a service-oriented architecture (SOA).

The properties and benefits of using a SOA such as web services is well suited for binding small modules that perform independent tasks within a highly heterogeneous e-business model. Web services can be easily wrapped around existing applications in your business model and plugged into different business processes.

For connecting to a large monolithic system that does not support the implementation of different flexible business processes, other approaches might be better suited, for example, to satisfy specialized features, such as performance or security.

The following business models are easily implemented by using an architecture including web services:

- **Business information**

  Sharing of information with consumers or other businesses. Web services can be used to expand the reach through such services as news streams, local weather reports, integrated travel planning, and intelligent agents.
- **Business integration**

  Providing transactional, fee-based services for customers. A global network of suppliers can be easily created. Web services can be implemented in auctions, e-marketplaces, and reservation systems.
- **Business process externalization**

> Web services can be used to model value chains by dynamically integrating processes to a new solution within an organizational unit or even with those of other e-businesses. This modeling can be achieved by dynamically linking internal applications to new partners and suppliers, to offer their services to complement internal services.

To see how these models are implemented using all aspects of Web services, see the web services scenario overview information to learn more about the story of a fictional online garden supply retailer named Plants by WebSphere and how this retailer incorporates the web services concept.

Web services

Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network.

The application server supports web services that are developed and implemented based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification. The application server supports the Java API for XML Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. The JAX-WS is a strategic programming model that simplifies application development through support of a standard, annotation-based model to develop web services applications and clients.

A typical web services scenario is a business application requesting a service from another existing application. The request is processed through a given web address using SOAP messages over a HTTP, Java Message Service (JMS) transport or invoked directly as Enterprise JavaBeans (EJB). The service receives the request, processes it, and returns a response. Examples of a simple web service include weather reports or getting stock quotes. The method call is synchronous, that is, the method waits until the result is available. Transaction web services, supporting quotes, business-to-business (B2B) or business-to-client (B2C) operations include airline reservations and purchase orders.

Web services can include the actual service or the client that accesses the service.

Web services are web applications that help improve the flexibility of your business processes by integrating with applications that otherwise do not communicate. The inner-library loan program at your local library is a good example of the web services concept and its evolution. The web service concept existed even before the term; the concept became widely accepted with the creation of the Internet. Before the Internet was created, you visited your library, searched the collections and checked out your books. If you did not find the book that you wanted, the librarian ran a search for you by computer or phone and located the book at a nearby library. The librarian ordered the book for you and you picked it up after it was delivered to your local library. By incorporating web services applications, you can streamline your library visit.

Now, you can search the local library collection and other local libraries at the same time. When other libraries provide your library with a web service to search their collection (the service might have been provided through Universal Description Discovery and Integration (UDDI), your results yield their resources. You might use another web service application to check out and send the book to your home. Using web services applications saves time and provides a convenience for you, as well as freeing the librarian to do other business tasks.

Web services reflect the service-oriented architecture (SOA) approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, web services applications provide components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

The key components of web services include:

- Web Services Description Language (WSDL)

  WSDL is the XML-based file that describes the web service. The web service request uses this file to bind to the service.
- SOAP

  SOAP is the XML-based protocol that the web service request uses to invoke the service.

For a more detailed scenario, see the web services scenario overview information to learn more about the story of a fictional online garden supply retailer named Plants by WebSphere, and how this retailer incorporated the web services concept.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Web Services for Java EE specification

The *Web Services for Java Platform, Enterprise Edition (Java EE)* specification defines the programming model and runtime architecture for implementing web services based on the Java language. Another name for the Web Services for Java EE specification is the Java Specification Requirements (JSR) 109. The specification includes open standards for developing and implementing web services.

The Web Services for Java EE specification is based on the Java EE technology and supports the Java API for XML Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) programming model for web services and clients in a manner that is interoperable and portable across application servers within environments that are scalable and secure. This specification is based on industry standards for web services, including Web Services Description Language (WSDL) and SOAP, and it describes the development and deployment of web services.

The application server supports the Web Services for Java EE specification, Version 1.3. This specification supports WSDL Version 1.1, SOAP Version 1.1 and SOAP Version 1.2.

You can integrate the Java EE technology with web services in a variety of ways. You can expose Java EE components as web services, for example, JavaBeans and enterprise beans. When you expose Java EE components as web services, clients that are written in Java code or existing web service clients that are not written in Java code can access these services. Java EE components can also act as web service clients.

The Web Services for Java EE specification is the preferred platform for Web-based programming because it provides open standards permitting different types of languages, operating systems and software to communicate seamlessly through the Internet.

For a Java application to act as web service client, a mapping between the WSDL file and the Java application must exist. For JAX-WS applications, the mapping is defined using annotations. You can optionally use the `webservices.xml` deployment descriptor to specify the location of the WSDL file and override the value defined in the @WebServices annotation. For JAX-RPC applications, you must define the JAX-RPC mapping file. To learn more about the mapping that is defined between the WSDL file and your web service application, see the JAX-WS specification or the JAX-RPC specification in the web services specifications and API documentation depending on the programming model used.

You can use a Java component to implement a Web service by specifying the component interface and binding information in the WSDL file and designing the application server infrastructure to accept the service request.

This entire process encompassed is based on the Web Services for Java EE specification.

The specification defines the `webservices.xml` deployment descriptor specifically for web services. The `webservices.xml` deployment descriptor file defines the set of web services that you can deploy in a Web Services for Java EE enabled container.

For JAX-WS web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

For example, if your service implementation class for your JAX-WS web service includes the @WebService annotation as follows:

```
@WebService(wsdlLocation="http://myhost.com/location/of/the/wsdl/ExampleService.wsdl")
```

and the `webservices.xml` specifies a different filename for the WSDL document as follows:

```
<webservices>
<webservice-description>
<webservice-description-name>ExampleService</webservice-description-name>
<wsdl-file>META-INF/wsdl/ExampleService.wsdl</wsdl-file>
...
</webservice-description>
</webservices>
```

then the value that is specified in the deployment descriptor, `META-INF/wsdl/ExampleService.wsdl` overrides the annotation value.

See section 5 of the Web Services for Java EE specification for details regarding the correlation between values specified in the web services deployment descriptor file and the attributes of the @WebService and the @WebServiceProvider annotations.

For JAX-RPC web services, you must define the deployment characteristics in the `webservices.xml` deployment descriptor file.

You are responsible for providing various elements to the deployment descriptor, including:

- Port name
- Port service implementation
- Port service endpoint interface
- Port WSDL definition
- Port QName
- MTOM/XOP support for JAX-WS web services
- Protocol binding for JAX-WS web services
- JAX-RPC mapping
- Handlers (optional)
- Servlet mapping (optional)

The Enterprise JavaBeans (EJB) 2.1 specification also states that for a web service developed from a session bean, the EJB deployment descriptor, `ejb-jar.xml`, must contain the service-endpoint element. The service-endpoint value must be the same as that stated in the `webservices.xml` deployment descriptor.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Artifacts used to develop web services

With *development artifacts*, you can develop an enterprise bean or a JavaBeans module into web services that are based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification.

To create a web service from an enterprise bean or from a JavaBeans module, the following files are added to the respective Java archive (JAR) file or web application archive (WAR) modules at assembly time:

- **Web Services Description Language (WSDL) Extensible Markup Language (XML) file**

  The WSDL XML file describes the web service that is implemented.
- **Service Endpoint Interface**

  A Service Endpoint Interface is the Java interface corresponding to the web service port type implemented. The Service Endpoint Interface is defined by the Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) web services run time that you are using.
- **webservices.xml**

  The `webservices.xml` file contains the Java EE deployment descriptor of the web service specifying how the web service is implemented. The `webservices.xml` file is defined in the Web Services for Java EE specification.

  For JAX-WS web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

  For JAX-RPC applications, deployment descriptors are required to specify how the web service is implemented.
- **ibm-webservices-bnd.xmi (JAX-RPC applications only)**

  This file contains WebSphere product-specific deployment information and is defined in the `ibm-webservices-bnd.xmi` deployment descriptor. assembly properties. See the JAX-RPC web services deployment descriptor settings information to learn more about this deployment descriptor.
- **Java API for XML-based remote procedure call (JAX-RPC) mapping file**

  The JAX-RPC mapping deployment descriptor specifies how Java elements are mapped to and from WSDL file elements.

The following files are added to an application client, enterprise beans or web module to permit a Web Services for Java Platform, Enterprise Edition (Java EE) client access to web services:

- **WSDL file**

  The WSDL file is provided by the web service implementer.
- **Java interfaces for the web service**

  The Java interfaces are generated from the WSDL file as specified by the JAX-WS or JAX-RPC specification. These bindings are the Service Endpoint Interface based on the WSDL port type, or the service interface, which is based on the WSDL service.
- **ibm-webservicesclient-bnd.xmi (JAX-RPC applications only)**

  This file contains WebSphere product-specific deployment information, such as security information for JAX-RPC applications. For JAX-WS applications, deployment descriptors are not supported and have been replaced by the use of annotations.
- **Other JAX-RPC binding files**

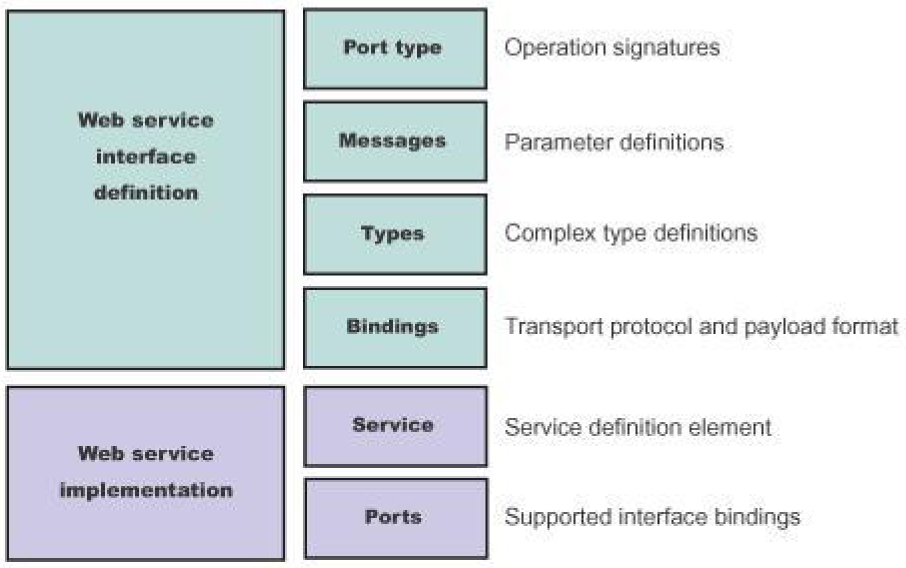
  Additional JAX-RPC binding files that support the client application in mapping SOAP to the Java language are generated from WSDL by the **`WSDL2Java`** command tool.

WSDL

Web Services Description Language (WSDL) is an Extensible Markup Language (XML)-based description language. This language was submitted to the World Wide Web Consortium (W3C) as the industry standard for describing web services. The power of WSDL is derived from two main architectural principles: the ability to describe a set of business operations and the ability to separate the description into two basic units. These units are a description of the operations and the details of how the operation and the information associated with it are packaged.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definitions of endpoints and messages are separated from their concrete network deployment or data format bindings. This separation supports the reuse of abstract definitions: messages, which are abstract descriptions of exchanged data, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Therefore, a WSDL document is composed of several elements.

The following is the structure of the information in a WSDL file:


Web service interface definition
Port type
Operation signatures
Messages
Parameter definitions
Types
Complex type definitions
Bindings
Transport protocol and payload format
Web service implementation
Service
Service definition element
Ports
Supported interface bindings

A WSDL file contains the following parts:

- **Web service interface definition**

  This part contains the elements and the namespaces.
- **Web service implementation**

  This part contains the definition of the service and ports.

A WSDL file describes a web service with the following elements:

portType

The description of the operations and associated messages. The portType element defines abstract operations.

```
<portType name="EightBall">
  <operation name="getAnswer">
    <input message="ebs:IngetAnswerRequest"/>
    <output message="ebs:OutgetAnswerResponse"/>
  </operation>
</portType>
```

message

The description of input and output parameters and return values.

```
<message name="IngetAnswerRequest">
  <part name="meth1_inType" type="ebs:questionType"/>
</message>
<message name="OutgetAnswerResponse">
  <part name="meth1_outType" type="ebs:answerType"/>
</message>
```

types

The schema for describing XML types used in the messages.

```
<types>
  <xsd:schema targetNamespace="...">
    <xsd:complexType name="questionType">
      <xsd:element name="question" type="string"/>
    </xsd:complexType>
    <xsd:complexType name="answerType">
    ...
</types>
```

binding

The bindings describe the protocol that is used to access a portType, as well as the data formats for the messages that are defined by a particular portType element.

```
<binding name="EightBallBinding" type="ebs:EightBall">
  <soap:binding style="rpc" transport="schemas.xmlsoap.org/soap/http">
  <operation name="ebs:getAnswer">
  <soap:operation soapAction="urn:EightBall"/>
    <input>
      <soap:body namespace="urn:EightBall" ... />
      ...
```

Service

The services and ports define the location of the Web service.

The service contains the web service name and a list of ports.

Ports

The ports contain the location of the web service and the binding used for service access.

```
<service name="EightBall">
  <port binding="ebs:EightBallBinding" name="EightBallPort">
    <soap:address location="localhost:8080/axis/EightBall"/>
  </port>
</service>
```

When creating Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) web services, you can use a bottom-up development approach when you start from JavaBeans or an enterprise bean, or you can use a top-down development approach when you start with an existing Web Services Description Language (WSDL) file.

When creating JAX-WS web services for this product, you can start with either a WSDL or an implementation bean class. If you start with an implementation bean class, then use the **`wsgen`** command line tool to generate all the web services server artifacts, including a WSDL if requested. If you start with a WSDL, then use the **`wsimport`** command line tool to generate all the web services artifacts for either the server or client side.

When creating JAX-RPC web services for this product, you must first have an implementation bean that includes a service endpoint interface. Then, you use the **`Java2WSDL`** command-line tool to create a WSDL file that defines the web services. If you are starting with the WSDL to generate the implementation bean class, run the **`WSDL2Java`** command line tool against the WSDL file to create Java APIs and deployment descriptor templates.

Multipart WSDL and WSDL publication

The product supports deployment of web services using a multipart Web Services Description Language (WSDL) file. In multipart WSDL files, an implementation WSDL file contains the wsdl:service. This implementation WSDL file imports an interface WSDL file, which contains the other WSDL constructs. This supports multiple web services using the same WSDL interface definition.

The <wsdl:import> element indicates a reference to another WSDL file. If the <wsdl:import> element location attribute does not contain a URL, that is, it contains only a file name, and does not begin with `http://`, `https://` or `file://`, the imported file must be located in the same directory and must not contain a relative path component. For example, if `META-INF/wsdl/A_Impl.wsdl` is in your module and contains the `<wsdl:import="A.wsdl" namespace="..."/>` import statement, the `A.wsdl` file must also be located in the module `META-INF/wsdl` directory.

It is recommended that you place all WSDL files in either the `META-INF/wsdl` directory, if you are using Enterprise JavaBeans (EJB), or the `WEB-INF/wsdl` directory, if you are using JavaBeans components, even if relative imports are located within the WSDL files. Otherwise, implications exist with the WSDL publication when you use a path like `<location="../interfaces/A_Interface.wsdl"namespace="..."/>`. Using a path like this example fails because the presence of the relative path, regardless of whether the file is located at that path or not. If the location is a web address, it must be readable at both deployment and server startup.

You can publish the files located in the `META-INF/wsdl` or the `WEB-INF/wsdl` directory through either a URL address or file, including WSDL or XML Schema Definition (XSD) files. For example, if the file referenced in the <wsdl-file> element of the `webservices.xml` deployment descriptor is located in the `META-INF/wsdl` or the `WEB-INF/wsdl` directory, it is publishable. If the files imported by the <wsdl-file> are located in the `wsdl/` directory or its subdirectory, they are publishable.

If the WSDL file referenced by the <wsdl-file> element is located in a directory other than `wsdl`, or its subdirectories, the file and its imported files, either WSDL or XSD files, which are in the same directory, are copied to the `wsdl` directory without modification when the application is installed. These types of files can also be published.

If the <wsdl-file> imports a file located in a different directory (a directory that is not `-INF/wsdl` or a subdirectory), the file is not copied to the `wsdl` directory and not available for publishing.

For JAX-WS web services, you can use an annotation to specify the location of the WSDL. Use the `@WebService` annotation with the `WSDLLocation` attribute. The `WSDLLocation` attribute is optional. If this attribute is not specified, then WSDL is generated and published from the information that is found in the web service classes. You can optionally specify the location of the WSDL file in the `webservices.xml` deployment descriptor. However, any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

SOAP

SOAP is a specification for the exchange of structured information in a decentralized, distributed environment. As such, it represents the main way of communication between the three key actors in a service oriented architecture (SOA): service provider, service requestor and service broker. The main goal of its design is to be simple and extensible. A SOAP message is used to request a web service.

SOAP 1.1

WebSphere Application Server follows the standards outlined in SOAP 1.1.

SOAP was submitted to the World Wide Web Consortium (W3C) as the basis of the Extensible Markup Language (XML) Protocol Working Group by several companies, including IBM and Lotus. This protocol consists of three parts:

- An *envelope* that defines a framework for describing message content and processing instructions.
- A set of *encoding rules* for expressing instances of application-defined data types.
- A *convention* for representing remote procedure calls and responses.

SOAP 1.1 is a protocol-independent transport and can be used in combination with a variety of protocols. In web services that are developed and implemented with WebSphere Application Server, SOAP is used in combination with HTTP, HTTP extension framework, and Java Message Service (JMS). SOAP is also operating-system independent and not tied to any programming language or component technology.

As long as the client can issue XML messages, it does not matter what technology is used to implement the client. Similarly, the service can be implemented in any language, as long as the service can process SOAP messages. Also, both server and client sides can reside on any suitable platform.

SOAP 1.2

The SOAP 1.2 specification is also a W3C recommendation, and WebSphere Application Server follows the standards that are outlined in SOAP 1.2. The SOAP 1.2 specification comes in three parts plus some assertions and a test collection:

- Part 0: Primer
- Part 1: Messaging Framework
- Part 2: Adjuncts
- Specification Assertions and Test Collection

SOAP 1.2 provides a more specific definition of the SOAP processing model, which removes many of the ambiguities that sometimes led to interoperability problems in the absence of the Web Services-Interoperability (WS-I) profiles. SOAP 1.2 should reduce the chances of interoperability issues with SOAP 1.2 implementations between different vendors.

Some of the more significant changes in the SOAP 1.2 specification include:

- The ability to now officially define other transport protocols other than the HTTP protocol as long as vendors conform to the binding framework that is defined in SOAP 1.2. While HTTP is ubiquitous, it is not as reliable of a transport as other things such as TCP/IP, MQ, and so forth.
- The fact that SOAP 1.2 is based on the XML Information Set (XML Infoset). The information set provides a way to describe the XML document using the XSD schema but does not necessarily serialize the document by using XML 1.0 serialization. SOAP 1.1 is based upon XML 1.0 serialization. The information set will make it easier to use other serialization formats such as a binary protocol format. You can use a binary protocol format shrink the message into a much more compact format where some of the verbose tagging information might not be required.

The Java API for XML Web Services (JAX-WS) standard introduces the ability to support both SOAP 1.1 as well as SOAP 1.2.

See the differences is SOAP versions information for additional differences between SOAP 1.1 and SOAP 1.2.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

SOAP with Attachments API for Java interface

The SOAP with Attachments API for Java (SAAJ) interface is used for SOAP messaging that provides a standard way to send XML documents over the Internet from a Java programming model. SAAJ is used to manipulate the SOAP message to the appropriate context as it traverses through the runtime environment.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

The Java API for XML-Based RPC (JAX-RPC) programming model supports SAAJ 1.2 to manipulate the XML.

The JAX-WS programming model supports SAAJ 1.2 and 1.3. SAAJ 1.3 includes support for SOAP 1.2 messages.

The differences in the SAAJ 1.2 and SAAJ 1.3 specification can be reviewed in the topic “Differences in SAAJ versions.”

How are messages used in web services?

Web services use XML technology to exchange messages. These messages conform to XML schema. When developing web services applications, there are limited XML APIs to work with, for example, Document Object Model (DOM). It is more efficient to manipulate the Java objects and have the serialization and deserialization completed during run time.

Web services uses SOAP messages to represent remote procedure calls between the client and the server. Typically, the SOAP message is deserialized into a series of Java value-type business objects that represent the parameters and return values. In addition, the Java programming model provides APIs that support applications and handlers to manipulate the SOAP message directly. Because there are a limited number of XML schema types that are supported by the programming models, the specification provides the SAAJ data model as an extension to manipulate the message.

To manipulate the XML schema types, you need to map the XML schema types to Java types with a custom data binder.

The SAAJ interface

The SAAJ-related classes are located in the javax.xml.soap package. SAAJ builds on the interfaces and abstract classes and many of the classes begin by invoking factory methods to create a factory such as SOAPConnectionFactory and SOAPFactory.

gotcha: If Java security is enabled, and permissions to read the jaxm.properties file is not granted, when a SOAPFactory instance is created through a call to javax.xml.soap.SOAPFactory.newInstance(), or a MessageFactory instance is created through a call to MessageFactory.newInstance(), a SecurityException exception occurs, and the following exception is written to the system log:

```
Permission:

      /opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties : access denied
(java.io.FilePermission /opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties
read)

Code:
     com.ibm.ws.wsfvt.test.binding.addr1.binder.AddressBinder
in  {file:/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/
ahp6405Node01Cell/DataBinding.ear/address1.war/WEB-INF/lib
/addressbinder1.jar}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
/opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties read)
.
```

The SOAPFactory ignores the exception, and continues on to the next means of determining which implementation to load. Therefore, you can ignore the log entry for this security exception.

Because this product uses the SOAPFactory to support other web services technologies, such as WS-Addressing (WS-A), WS-Atomic Transaction (WS-AT), and WS-Notification, you can ignore this SecurityException in any web services application where Java security is enabled.

The most commonly used classes are:

- SOAPMessage: Contains the message, both the XML and non-XML parts
- SOAPHeader: Represents the SOAP header XML element
- SOAPBody: Represents the SOAP body XML element
- SOAPElement: Represents the other elements in the SOAP message

Other parts of the SAAJ interface include:

- MessageContext: Contains a SOAP message and related properties
- AttachmentPart: Represents a binary attachment
- SOAPPart: Represents the XML part of the message
- SOAPEnvelope: Represents the SOAP envelope XML element
- SOAPFault: Represents the SOAP fault XML element

The primary interface in the SAAJ model is javax.xml.soap.SOAPElement, also referred to as SOAPElement. Using this model, applications can process an SAAJ model that uses pre-existing DOM code. It is also easier to convert pre-existing DOM objects to SAAJ objects.

Messages created using the SAAJ interface follow SOAP standards. A SOAP message is represented in the SAAJ model as a javax.xml.soap.SOAPMessage object. The XML content of the message is represented by a javax.xml.soap.SOAPPart object. Each SOAP part has a SOAP envelope. This envelope is represented by the SAAJ javax.xml.SOAPEnvelope object. The SOAP specification defines various elements that reside in the SOAP envelope; SAAJ defines objects for the various elements in the SOAP envelope.

The SOAP message can also contain non-XML data that is called attachments. These attachments are represented by SAAJ AttachmentPart objects that are accessible from the SOAPMessage object.

A number of reasons exist as to why handlers and applications use the generic SOAPElement API instead of a tightly bound mapping:

- The web service might be a conduit to another web service. In this case, the SOAP message is only forwarded.
- The web service might manipulate the message using a different data model, for example a Service Data Object (SDO). It is easier to convert the message from a SAAJ DOM to a different data model.
- A handler, for example, a digital signature validation handler, might want to manipulate the message generically.

You might need to go a step further to map your XML schema types, because the SOAPElement interface is not always the best alternative for legacy systems. In this case you might want to use a generic programming model, such as SDO, which is more appropriate for data-centric applications.

The XML schema can be configured to include a custom data binding that pairs the SDO or data object with the Java object. For example, the run time renders an incoming SOAP message into a SOAPElement interface and passes it to the customer data binder for more processing. If the incoming message contains an SDO, the run time recognizes the data object code, queries its type mapping to locate a custom binder, and builds the SOAPElement interface that represents the SDO code. The SOAPElement is passed to the SDOCustomBinder.

See information on custom data binders to learn more about the process of developing applications with SOAPElement, SDO and custom binders.

Starting in WebSphere Application Server Version 8, the `SOAPMessage.getSOAPHeader` and `getSOAPBody` methods now throw a `SOAPException` if there is no corresponding element in the message. A System property is provided to revert the behavior to return null rather than throw an exception. The property is defined in `com.ibm.websphere.webservices.soap.IBMSOAPMessage.ENABLE_LEGACY_GETSOAP_BEHAVIOR` as a String value of `com.ibm.websphere.webservices.soap.enable.legacy.get.behavior`. The default value of the property is null which is interpreted as `false`. To revert the behavior to return a null, set the property to the String value `true`. The previous behavior of returning null is not compliant with the specification.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Differences in SAAJ versions

The SOAP with Attachments API for Java (SAAJ) interface Version 1.3 expands the support of SOAP 1.2 messages in a web services environment. There are several differences between SAAJ 1.2 and SAAJ 1.3 that are presented in this topic.

In a typical web services environment, you rely on the underlying code that is based on Java standards to translate a set of Java objects. The SAAJ interface provides APIs to read, write, send and receive SOAP messages, and advocates binary content sent as an attachment to a SOAP message.

SAAJ 1.3 aligns with SOAP 1.1 and SOAP 1.2 messages and is supported by the Java API for XML Web Services (JAX-WS) programming model and the Java API for XML-Based RPC (JAX-RPC) programming model. SAAJ 1.2 works with SOAP 1.1 messages only.

If you migrate your code from SOAP 1.1 to SOAP 1.2, you can continue to use your existing SOAP 1.1 code, if the message is a SOAP 1.2 message. If you upgrade your base code to use SAAJ 1.3, then you can continue to use the existing code that operates on a SOAP 1.1 message. An example of these differences is in SOAP 1.1, where the human readable text of a fault is stored in the faultString element. In SOAP 1.2, the human readable text is stored in the Reason element. Your code might look like the following example:

```
String text = soapFault.getFaultString();
```

The getFaultString () returns the faultString value if the message is based on SOAP 1.1. If you are using SOAP 1.2, the getFaultString () returns the Reason value. In addition, the SAAJ 1.3 interface provides a new method, getReasonText (Locale), that gets a specific Reason value. The getReasonText (Locale) method returns a documented exception if the message is based on SOAP 1.1. The SAAJ 1.3 interface supports existing code to process both SOAP 1.1 and SOAP 1.2 messages.

Other differences between SAAJ 1.2 and SAAJ 1.3 are in the following list:

- SAAJMetaFactory interface

  The SAAJMetaFactory SPI is introduced to support creating SOAP factory classes in a single place.

- SAAJResult class

  The SAAJResult object acts as a holder for the results of a Java API for XML Processing (JAX-P) transformation or a Java Architecture for XML Binding (JAXB) marshalling, in the SAAJ tree. The SAAJResult class is introduced for improved usability when transformation results are expected to be a valid SAAJ tree.
- Overloaded methods that accept a QName instead of a Name

  It is preferred that a QName represents an XML-qualified name. Therefore, overloaded methods are introduced in all of the SAAJ APIs, where a corresponding method accepts a javax.xml.soap.Name name as an argument.
- New methods in AttachmentPart, SOAPBody and SOAPElement interfaces and classes

  Use these new methods to assist you when you are working with the new SOAP features.
- SOAPPart is now a javax.xml.soap.Node method.

  The SOAPPart object is now also considered to be a SOAP node method.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Message Transmission Optimization Mechanism

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

There are many reasons why you might want to send binary attachments, such as images or files, along with a web services request. There are ways to accomplish this, such as:

- Encoding with base64 inline in the SOAP payload. However, encoding inline tends to enlarge the size of the SOAP message. Note that base64 encoding might double the size of the binary data.
- Encoding the messages by using SOAP with Attachments (SwA) and to follow the Web Services Interoperability Organization (WS-I) Attachments Profile. WebSphere Application Server currently supports this method.
- Providing optimization of binary message transportation by using XML-binary Optimized Packaging (XOP). Optimization is available only for binary data or content. MTOM uses XOP in the context of SOAP and MIME over HTTP.

XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that just happens to look like a MIME multipart/related package, with an XML documents as the root part. That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that is associated with encoding. Encoding is the only way a binary data can fit directly into an XML world.

If MTOM mapping generation is disabled, then XOP is disabled. If XOP is disabled, the binary data are not sent by using MIME attachments. Instead, the binary data is base64 encoded as usual.

The MTOM specification is defined in three different parts:

- An abstract feature for optimized transmission or wire format for SOAP messages. This feature is abstract in the sense that the description of the optimization technique as well as the behavior of the SOAP processors at sender, receiver and intermediaries is generic and does not include any references to technologies such as MIME, HTTP, and so forth. The optimization technique centers around ensuring a SOAP envelope Infoset view for the SOAP processors while encoding selectively certain contents of the SOAP Envelope Infoset that are accessible as canonical lexical representation of the xs:base64Binary data type.

Implementing these abstract features requires concrete specification of two aspects: the optimized wire format and how the optimized wire format flows on a particular transport

- The second part of the MTOM specification addresses the serialization aspect and depends normatively upon MIME Multipart/Related XOP packaging. The serialization aspect is where MTOM relates to XOP.
- As a concrete SOAP HTTP binding level feature, MTOM expands upon the serialization. This part describes how HTTP binding can be used to transport the XOP packages that are holding the SOAP MTOM messages. This part also puts some restrictions on the possible serializations of the SOAP MTOM messages as XOP packages, such as use of XML 1.0 only.

The Java API for XML Web Services (JAX-WS) adds support for sending binary data attachments using MTOM. JAX-WS is the centerpiece of a newly re-architected API stack for web service that includes JAX-WS 2.0, JAXB 2.0, and SAAJ 1.3. The API stack is sometimes referred to as the integrated stack. JAX-WS is designed to take the place of JAX-RPC in web services and web applications.

Attachment approach

Attachment by value or by reference has been the widely accepted technique for handling opaque data in XML-formatted messages.

- **By value** is when the opaque data content is embedded as an element or as an attribute by using either base64 or hexadecimal text encoding approach, which is codified in the XML schema as data types xs:base64Binary and xs:hexBinary, respectively.
- **By reference** is when the opaque data content is referenced externally as element or as attribute by using a URI, which is codified in the XML schema as data type xs:anyURI.

The use of either of these two techniques has its advantages and disadvantages. MTOM is the specification that is focused on resolving these inherent attachments problems.

A different standard is defined by World Wide Web Consortium (W3C) and is called SOAP with Attachments (SwA). SwA was developed as a way to package SOAP messages with attachments. Because some vendors do not support SwA, SwA can be replaced by the more powerful MTOM and XOP mechanisms. SwA and MTOM are conceptually similar, and both encode binary data as a MIME attachment in a MIME document. Using MIME attachments improves the performance of large binary payloads transport.

Additional differences between SwA and MTOM include:

- MTOM uses a standard called XOP, which defines a XOP reference that exists within the SOAP payload. This reference points to the MIME attachment that contains the binary data.
- With MTOM, the XOP reference logically includes the binary data into the XML Information Set (Infoset). With SwA, the href points to data that is not only physically outside the XML document but is not logically included within its Infoset.
- With MTOM, binary attachments can be logically signed as if they were part of the SOAP XML document.
- In additional to IBM, Microsoft .NET supports MTOM, which eliminates some of the interoperability problems found with SwA. Interoperability was treated as the main goal when the co-submitters discussed the suggested modifications.

The MTOM attachment approach takes advantage of the SOAP infrastructure while also gaining transport efficiencies that are provided by SOAP with Attachment (SwA) solution.

SOAP 1.2 and SOAP 1.1

SOAP 1.1 is based on the XML specification. Likely, the SOAP 1.1 implementation will continue to exist for a few years. For those who are still running SOAP 1.1, there is now an interoperable way to use MTOM for attachments support. SAP, Oracle, Microsoft, and IBM have submitted a SOAP 1.1 Binding for MTOM 1.0 specification to W3C, which defines how MTOM can be used with SOAP 1.1 payloads. The specification details the necessary modifications to the SOAP MTOM and XOP specifications that are necessary to successfully use these technologies with SOAP 1.1. See the specification to learn more details.

MTOM is a SOAP Version 1.2 feature, which is based on the Infoset. See the XML information set documentation to learn more.

Without MTOM, the data is encoded in whatever format is described in the schema (base64 or hex) and then is contained in the XML document. The following example shows a SOAP message with an <xsd:base64Binary> element:

```
... other transport headers ...
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
      <input>
        <imageData>R0lGODl  ... more base64 encoded data ...  KTJk8giAAA7</imageData>
      </input>
    </sendImage>
  </soapenv:Body>
</soapenv:Envelope>
```

When MTOM is enabled, the binary data that represents the attachment is included as a MIME attachment to the SOAP message. The following example shows an MTOM-enabled SOAP message with attachment data:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;
type="application/xop+xml"; start="<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>";
start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
      <input>
        <imageData>
          <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
               href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
        </imageData>
      </input>
    </sendImage>
   </soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

XML-binary Optimized Packaging:

XML-binary Optimized Packaging (XOP) specification was standardized by the World Wide Web (W3C) on January 25, 2005. SOAP Message Transmission Optimization Mechanism (MTOM) uses XOP in the context of SOAP and MIME over HTTP.

XML is widely used for data transfer. XML is a popular format for exchanging well-formed documents because it is plain text, human-readable, and structured. For example, SOAP messaging in web services is based on XML (or is based on XML Infoset with SOAP 1.2). People want to leverage legacy formats like PDF, GIF, JPEG and similar things, while still using an XML model. The desire to integrate XML with pre-existing data formats has been a long-standing and persistent issue for the XML community. Users often want to leverage the structured, extensible markup conventions of XML without abandoning existing data formats that do not readily adhere to XML 1.0 syntax.

As SOAP messaging in web services becomes more widespread, the next step is how to send non-text based data, such as images and workflow data, along with your message. For example, you might want to send a scanned document in .jpeg format between two applications. The question becomes whether this data can be understood between the various applications.

Much of the value of XML and web services resides in the ability to use generic XML tools to work with content. Many XML tools and standards for describing and manipulating XML (such as parsers, XPath, XQuery, XSLT, XML encryption and digital signature and XML schema) are not designed to work with non-text data, such as images. These XML tools do not work with non-XML content; these tools require text. The challenge is how non-text data (also called *binary data*) can be embedded or attached with XML. In other words, a way to attach a binary file to a SOAP message is needed.

Encoding is the only way binary data can fit directly into an XML world. Normally, you can embed binary data in an XML document by encoding it as text using Base 64. Base 64 is a serialization that has existed for some time, can be easily implemented out of the box, and has interoperability across platforms. The xsi:base64binary datatype supports this serialization in the XML Schema. Base 64 encodes your binary data into a textual representation that can squeeze into an XML document. Base 64 takes your binary data and translates it into a series of ASCII characters by encoding three octets at a time. Because each octet consists of eight bits, representing them as four printable characters in the ASCII standard, it uses 64 ASCII characters to represent the binary. All platforms can decode and encode using this convention. 6-bit ASCII is widely supported, and no special characters need to be dealt with. However, there is a performance impact for larger messages.

For applications that require speedy operation, Base 64 might not be the solution. If you want to index into such content, query it, transform it, encrypt it, sign it, or describe it, you need to use a different mechanism.

The first attachment specification known as SOAP with Attachments (SwA) was developed. The basic idea of SwA is that the binary message part (the attachment) is thought of as a Multipurpose Internet Mail Extensions (MIME) attachment. MIME is a widely implemented specification for formatting non-ASCII mail message attachments. SwA specifies that the SOAP body can contain a reference to the MIME message part (the attachment) simply by using a URI. The binary part is attached by a reference.

A few disadvantages of SwA include:

- SwA fails in its usability or interoperability. The SOAP infrastructure was created around the SOAP envelope, which didn't work well for attachments. An attachment using SwA means that two data models are used in one message. These two data models do not operate with existing XML technology.
- SwA does not work with the composable character of SOAP. Basically standards, such as WS-Security, were not written to work with attachments. WS-Security needs to work on all the data that needs to be digitally signed or encrypted, and that means all the data in the attachment also. But if it cannot access it, then it will not work and the signature is effectively invalid.

Often, users want to leave their existing non-XML formats as is, to be treated as opaque sequences of octets by XML tools and infrastructure. Such an approach permits widely used formats such as .jpeg and .wav to peacefully coexist with XML. XOP makes it a bit more realistic to use base64-encoded data. At the current time, XOP only permits base64-encoded data to be optimized.

Using XOP means that instances of XML-type base64Binary, if enabled, are transported by using MIME attachments. If XOP is in use, the implementation can automatically encode it. XOP maintains the data model of the XML message because the attachment is treated as base64-encoded data. If an XML stack understands XOP encoding, your application does not need to be changed at all. For example, when it wants to access a .jpeg picture, it can get the character value of the content as a base64-encoded string.

XOP gives people a way to think about MIME messages in a data exchange that they are comfortable with and already use for a lot of other data. The XOP format uses multipart MIME to enable raw binary data to be included into an XML 1.0 document without resorting to base64 encoding.

A companion specification, SOAP Message Transmission Optimization Method (MTOM) then specifies how to bind this format to SOAP. The XOP and MTOM standards should enhance SOAP 1.2 performance. XOP and MTOM together provide the preferred approach for mixing binary data with text-based XML. Coupled together, MTOM and XOP enables us to select what parts of the message need to be sent over the wire as binary while still maintaining the Infoset. These standards enable the attachment of binary data outside of the SOAP envelope as a message part. However, unlike SwA, the binary data is treated very much as it was within the SOAP envelope, meaning one Infoset.

XOP defines a serialization mechanism for XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but applicable to any XML Infoset and any packaging mechanism. On the other hand, XML is not a good general-purpose packaging mechanism.

An XOP package is created by placing a serialization of the XML Infoset inside of an extensible packaging format (such a MIME). Note that XOP does reuse MIME for the actual packaging on the wire. Then, selected portions of its content that are base64-encoded binary data are extracted and re-encoded, meaning the data is decoded from base64 and placed into the package. The locations of those selected portions are marked in the XML with a special element that links to the packaged data by using URIs.

The SOAP processing engines performs a temporary Base 64 encoding of the binary data just before the message hits the wire. This temporary encoding enables the SOAP processor to work on the Base 64 data; for example, enabling a WS-Signature of the data to be taken and placed into the header. There is no need for expensive decoding at the other end, and the process works in reverse.

Implementations of MTOM and XOP are available in Java (JAX-WS).

This example shows an XML Infoset prior to XOP processing (SOAP):

```
<soap:Envelope
      xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
      xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo xmlmime:contentType='image/png'>/aWKKapGGyQ=</m:photo>
      <m:sig xmlmime:contentType='application/pkcs7-signature'>Faa7vROi2VQ=</m:sig>
    </m:data>
  </soap:Body>
</soap:Envelope>
```

This example shows an XML Infoset that is serialized as a XOP package (SOAP)

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
    start="<mymessage.xml@example.org>";
    startinfo="application/soap+xml; action=\"ProcessData\""
Content-Description: A SOAP message with my pic and sig in it

--MIME_boundary
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="application/soap+xml; action=\"ProcessData\""
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<soap:Envelope
    xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
    xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo
  xmlmime:contentType='image/png'><xop:Include
    xmlns:xop='http://www.w3.org/2004/08/xop/include'
    href='cid:http://example.org/me.png'/></m:photo>
      <m:sig
  xmlmime:contentType='application/pkcs7-signature'><xop:Include
    xmlns:xop='http://www.w3.org/2004/08/xop/include'
    href='cid:http://example.org/my.hsh'/></m:sig>
    </m:data>
  </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/me.png>

// binary octets for png

--MIME_boundary
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/my.hsh>

// binary octets for signature

--MIME_boundary--
```

XML information set:

XML Information Set (Infoset) is a World Wide Web Consortium (W3C) specification, dated February 4, 2004. An XML information set is an abstract model of the information that is stored in an XML document. The information set establishes a separation between data and information in a way that suits most common uses of XML. Several of the concrete XML data models are defined by referring to XML information set items and their properties.

Whereas an *XML information set* is an abstract model of the information that is stored in an XML document, an *information item* is an abstract representation of some component of an XML document. SOAP Version 1.2 makes use of this abstraction to define the information in a SOAP message without ever referring to XML Version 1.x. The SOAP HTTP binding specifically permits alternative media types that provide for, as a minimum, the transfer of the SOAP XML Infoset.

SOAP Message Transmission Optimization Mechanism (MTOM) describes SOAP 1.2 constructs in terms of information items whereas SOAP 1.1 is defined in terms of XML elements. MTOM enables SOAP bindings to optimize the transmission or wire format (or both) of a SOAP message by selectively encoding portions of the message while still presenting an XML information set to the SOAP application. The SOAP 1.2 attribute is now in the SOAP namespace. The XML information sets require the support of XML namespaces. The core XML recommendation does not require the support of XML namespaces; however namespaces are required to support the XML schema.

The XML information set does not require or favor a specific interface or class of interfaces. The XML information set specification presents the information set as a tree for the sake of clarity and simplicity, but there is no requirement that the XML information set be made available through a tree structure. Other types of interfaces, including but not limited to event-based and query-based interfaces, are also capable of providing information conforming to the information set. As long as the information in the information set is made available to XML applications in one way or another, the requirements of the XML information set are satisfied.

The XML information set provides a set of definitions to be used in other specifications that refer to the information in a well-formed XML document. For any given XML document, there are a number of corresponding information sets.

- A unique minimal information set consisting of the core properties of the core items and nothing else.
- A unique maximal information set consisting of all the core and all the peripheral items with all the peripheral properties, and one for every combination of present and absent peripheral items and properties in between. The in-between information sets must be fully consistent with the maximal information set.

Information set items

The XML information set is a description of the information that is available in a well-formed XML document, and it describes an abstract data model of an XML document in terms of a set of information set items. An information item is an abstract description of some part of an XML document, and each information item has a set of associated named properties. All other information items are accessible from the properties of the document information item, either directly or indirectly through the properties of other information items.

Guidelines for using information set items include:

- There is no requirement for an XML document to be valid in order to have an information set.
- An XML document has an information set if it satisfies the namespace constraints.
- An XML document has an information set if it is well-formed
- Only one document information item is permitted in the information set.
- An information set for an XML document consists of two or more information items.
- The information set for any well-formed XML document will contain at least the minimum information items: one document information item and one element information item.
- Each information item has a set of associated properties, some of which are core and some of which are peripheral.

An information set can contain up to eleven different types of information items:

- Document information item
- Element information items
- Attribute information items
- Processing instruction information items
- Unexpanded entity reference information items
- Character information items
- Comment information items
- The Document Type Declaration (DTD) information item
- Unparsed entity information items
- Notation information items
- Namespace information items

Note that the information set of the XML document might not be a complete list of all information items.

Certain kinds of invalidity affect the values assigned to some properties. Entities, notations, elements and attributes can be undeclared. You can have multiple declarations for notations and elements. Multiple declarations are valid for entities and attributes. An ID can be undefined or multiply defined. Such cases are noted where relevant in the information item definitions in the XML Information Set specification.

Syntax

The XML information set uses a square-bracket syntax, meaning the property names are shown in square brackets. For example, the document information item has the following properties:

Table 53. XML information syntax. Specifies the syntax for property names for an XML document information item.

Property	Description
[children]	An ordered list of child information items, in document order.
[document element]	The element information item corresponding to the document element.
[notations]	An unordered set of notation information items, one for each notation declared in the DTD. If any notation is multiply declared, this property has no value.
[unparsed entities]	An unordered set of unparsed entity information items, one for each unparsed entity declared in the DTD.
[base URI]	The base URI of the document entity.
[character encoding scheme]	The name of the character encoding scheme in which the document entity is expressed.
[standalone]	An indication of the stand-alone status of the document, either yes or no. This property is derived from the optional standalone document declaration in the XML declaration at the beginning of the document entity, and has no value if there is no standalone document declaration.
[version]	A string representing the XML version of the document. This property is derived from the XML declaration optionally present at the beginning of the document entity, and has no value if there is no XML declaration.

Table 53. XML information syntax (continued). Specifies the syntax for property names for an XML document information item.

Property	Description
[all declarations processed]	This property is not strictly speaking part of the information set of the document. Rather it is an indication of whether the processor has read the complete DTD. Its value is a boolean. If it is false, then certain properties (indicated in their descriptions later in this section) might be unknown. If it is true, those properties are never unknown.

All information sets are understood to describe the XML document with all entity references already expanded; that is, represented by the information items corresponding to their replacement text. In the case that an entity reference cannot be expanded, because an XML processor has not read its declaration or its value, explicit provision is made for representing such a reference in the information set.

Differences in SOAP versions

Both SOAP Version 1.1 and SOAP Version 1.2 are World Wide Web Consortium (W3C) standards. Web services can be deployed that support not only SOAP 1.1 but also support SOAP 1.2. Some changes from SOAP 1.1 that were made to the SOAP 1.2 specification are significant, while other changes are minor.

The SOAP 1.2 specification introduces several changes to SOAP 1.1. This information is not intended to be an in-depth description of all the new or changed features for SOAP 1.1 and SOAP 1.2. Instead, this information highlights some of the more important differences between the current versions of SOAP.

The changes to the SOAP 1.2 specification that are significant include the following updates:

- SOAP 1.1 is based on XML 1.0. SOAP 1.2 is based on XML Information Set (XML Infoset).

  The XML information set (infoset) provides a way to describe the XML document with XSD schema. However, the infoset does not necessarily serialize the document with XML 1.0 serialization on which SOAP 1.1 is based.. This new way to describe the XML document helps reveal other serialization formats, such as a binary protocol format. You can use the binary protocol format to compact the message into a compact format, where some of the verbose tagging information might not be required.

  In SOAP 1.2 , you can use the specification of a binding to an underlying protocol to determine which XML serialization is used in the underlying protocol data units. The HTTP binding that is specified in SOAP 1.2 - Part 2 uses XML 1.0 as the serialization of the SOAP message infoset.
- SOAP 1.2 provides the ability to officially define transport protocols, other than using HTTP, as long as the vendor conforms to the binding framework that is defined in SOAP 1.2. While HTTP is ubiquitous, it is not as reliable as other transports including TCP/IP and MQ.
- SOAP 1.2 provides a more specific definition of the SOAP processing model that removes many of the ambiguities that might lead to interoperability errors in the absence of the Web Services-Interoperability (WS-I) profiles. The goal is to significantly reduce the chances of interoperability issues between different vendors that use SOAP 1.2 implementations.
- SOAP with Attachments API for Java (SAAJ) can also stand alone as a simple mechanism to issue SOAP requests. A major change to the SAAJ specification is the ability to represent SOAP 1.1 messages and the additional SOAP 1.2 formatted messages. For example, SAAJ Version 1.3 introduces a new set of constants and methods that are more conducive to SOAP 1.2 (such as getRole(), getRelay()) on SOAP header elements. There are also additional methods on the factories for SAAJ to create appropriate SOAP 1.1 or SOAP 1.2 messages.
- The XML namespaces for the envelope and encoding schemas have changed for SOAP 1.2. These changes distinguish SOAP processors from SOAP 1.1 and SOAP 1.2 messages and supports changes in the SOAP schema, without affecting existing implementations.
- Java Architecture for XML Web Services (JAX-WS) introduces the ability to support both SOAP 1.1 and SOAP 1.2. Because JAX-RPC introduced a requirement to manipulate a SOAP message as it traversed through the run time, there became a need to represent this message in its appropriate SOAP context. In JAX-WS, a number of additional enhancements result from the support for SAAJ 1.3.
- The Web Services Description Language (WSDL) Version 1.1 specification does not discuss SOAP 1.2. SOAP 1.2 is discussed in the draft versions of WSDL 2.0. WSDL Version 1.1 only defines how to render

a SOAP 1.1 payload in a WSDL 1.1 document. To resolve how to represent SOAP 1.2-based services, there is another W3C document that defines how to define a SOAP 1.2 payload within a WSDL 1.1 document. Read about WSDL 1.1 binding extensions for SOAP 1.2.

- SOAP 1.1 is a single document. The SOAP 1.2 specification is organized in the following parts:
  - Part 0 is a non-normative introduction to SOAP.
  - Part 1 describes the structure of SOAP messages, the SOAP processing model and a framework for binding SOAP to underlying protocols. Conformant SOAP implementations must implement everything in Part 1.
  - Part 2 describes optional add-ins to the core of SOAP including a data model and encoding, an RPC convention and a binding to HTTP. Conformant SOAP implementations might implement any of the add-ins in Part 2. However, if add-ins are implemented, they must conform to the relevant parts of the specification.

  A fourth document is the Specification Assertions and Test Collection

SOAP 1.2 has a number of changes in syntax and provides additional, clarified semantics from those that are described in SOAP 1.1. The SOAP 1.2 Primer document lists and describes these syntax changes.

JAX-WS

Java API for XML-Based Web Services (JAX-WS) is the next generation web services programming model complimenting the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, development of web services and clients is simplified with more platform independence for Java applications by the use of dynamic proxies and Java annotations.

JAX-WS is a programming model that simplifies application development through support of a standard, annotation-based model to develop web service applications and clients. The JAX-WS technology strategically aligns itself with the current industry trend towards a more document-centric messaging model and replaces the remote procedure call programming model as defined by JAX-RPC. While the JAX-RPC programming model and applications are still supported by this product, JAX-RPC has limitations and does not support various complex document-centric services. JAX-WS is the strategic programming model for developing web services and is a required part of the Java Platform, Enterprise Edition 6 (Java EE 6). JAX-WS is also known as JSR 224.

Version 8.0 supports the JAX-WS Version 2.2 and Web Services for Java EE (JSR 109) Version 1.3 specifications.

The JAX-WS 2.2 specification supersedes and includes functions within the JAX-WS 2.1 specification. JAX-WS 2.2 adds client-side support for using WebServiceFeature-related annotations such as @MTOM, @Addressing, and the @RespectBinding annotations. JAX-WS 2.1 had previously added support for these annotations on the server. There is also now the ability to enable and configure WS-Addressing support on a client or service by adding WS-Policy assertions into the WSDL document. In addition, the Web Services for Java EE 1.3 specification introduces support for these WebServiceFeature-related annotations, as well as support for using deployment descriptor elements to configure these features on both the client and server. JAX-WS 2.2 requires Java Architecture for XML Binding (JAXB) Version 2.2 for data binding.

The implementation of the JAX-WS programming standard provides the following enhancements for developing web services and clients:

- **Enhanced platform independence for Java applications.**

  Using JAX-WS APIs, development of web services and clients is simplified with enhanced platform independence for Java applications. JAX-WS takes advantage of the dynamic proxy mechanism to provide a formal delegation model with a pluggable provider. This is an enhancement over JAX-RPC, which relies on the generation of vendor-specific stubs for invocation.
- **Annotations**

JAX-WS provides support for annotating Java classes with metadata to indicate that the Java class is a Web service. JAX-WS supports the use of annotations based on the Metadata Facility for the Java Programming Language (JSR 175) specification, the Web Services Metadata for the Java Platform (JSR 181) specification and annotations defined by the JAX-WS 2.2 specification. Using annotations within the Java source and within the Java class simplifies development of web services. Use annotations to define information that is typically specified in deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL files into the source artifacts.

For example, you can embed a simple @WebService tag in the Java source to expose the bean as a web service.

```
@WebService

public class QuoteBean implements StockQuote {

       public float getQuote(String sym) { ... }

}
```

The @WebService annotation tells the server runtime environment to expose all public methods on that bean as a web service. Additional levels of granularity can be controlled by adding additional annotations on individual methods or parameters. Using annotations makes it much easier to expose Java artifacts as web services. In addition, as artifacts are created from using some of the top-down mapping tools starting from a WSDL file, annotations are included within the source and Java classes as a way of capturing the metadata along with the source files.

Using annotations also improves the development of web services within a team structure because you do not need to define every web service in a single or common deployment descriptor as required with JAX-RPC web services. Taking advantage of annotations with JAX-WS web services enables parallel development of the service and the required metadata.

For JAX-WS web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

For example, if your service implementation class for your JAX-WS web service includes the following:

– the @WebService annotation:

```
@WebService(wsdlLocation="http://myhost.com/location/of/the/wsdl/ExampleService.wsdl")
```

– the `webservices.xml` file specifies a different file name for the WSDL document as follows:

```
<webservices>
<webservice-description>
<webservice-description-name>ExampleService</webservice-description-name>
<wsdl-file>META-INF/wsdl/ExampleService.wsdl</wsdl-file>
...
</webservice-description>
</webservices>
```

In this case, the value that is specified in the deployment descriptor, `META-INF/wsdl/ExampleService.wsdl` overrides the annotation value.

- **Invoking web services asynchronously**

  With JAX-WS, Web services are called both synchronously and asynchronously. JAX-WS adds support for both a polling and callback mechanism when calling web services asynchronously. Using a polling model, a client can issue a request, get a response object back, which is polled to determine if the server has responded. When the server responds, the actual response is retrieved. Using the callback model, the client provides a callback handler to accept and process the inbound response object. Both the polling and callback models enable the client to focus on continuing to process work without waiting for a response to return, while providing for a more dynamic and efficient model to invoke Web services.

  For example, a web service interface might have methods for both synchronous and asynchronous requests. Asynchronous requests are identified in bold in the following example:

```
@WebService
public interface CreditRatingService {
      // sync operation
      Score      getCreditScore(Customer customer);
      // async operation with polling
      Response<Score> getCreditScoreAsync(Customer customer);
      // async operation with callback
      Future<?> getCreditScoreAsync(Customer customer,
         AsyncHandler<Score> handler);
}
```

The asynchronous invocation that uses the callback mechanism requires an additional input by the client programmer. The callback is an object that contains the application code that is run when an asynchronous response is received. Use the following code example to invoke an asynchronous callback handler:

```
CreditRatingService svc = ...;

Future<?> invocation = svc.getCreditScoreAsync(customerFred,
 new AsyncHandler<Score>() {
    public void handleResponse (
        Response<Score> response)
      {
        Score score = response.get();
        // do work here...
      }
   }
);
```

Use the following code example to invoke an asynchronous polling client:

```
CreditRatingService svc = ...;
Response<Score> response = svc.getCreditScoreAsync(customerFred);

while (!response.isDone()) {
  // Complete an action while we wait.
}

// No cast needed, because of generics.
Score score = response.get();
```

- **Using resource injection**

  JAX-WS supports resource injection to further simplify development of web services. JAX-WS uses this key feature of Java EE 5 to shift the burden of creating and initializing common resources in a Java runtime environment from your web service application to the application container environment, itself. JAX-WS provides support for a subset of annotations that are defined in JSR-250 for resource injection and application life cycle in its runtime environment.

  The application server also supports the usage of the @Resource or @WebServiceRef annotation to declare JAX-WS managed clients and to request injection of JAX-WS services and ports. When either of these annotations are used on a field or method, they result in injection of a JAX-WS service or port instance. The usage of these annotations also results in the type specified by the annotation being bound into the JNDI namespace.

  The @Resource annotation is defined by the JSR-250, Common Annotations specification that is included in Java Platform, Enterprise Edition 5 (Java EE 5). By placing the @Resource annotation on a variable of type `javax.xml.ws.WebServiceContext` within a service endpoint implementation class, you can request a resource injection and collect the `javax.xml.ws.WebServiceContext` interface related to that particular endpoint invocation. From the WebServiceContext interface, you can collect the `MessageContext` for the request associated with the particular method call using the `getMessageContext()` method.

  The @WebServiceRef annotation is defined by the JAX-WS specification.

  The following example illustrates using the @Resource and @WebServiceRef annotations for resource injection:

```
@WebService
public class MyService {

    @Resource
    private WebServiceContext ctx;

    @Resource
    private SampleService svc;

    @WebServiceRef
    private SamplePort port;

    public String echo (String input) {
        ...
    }

}
```

Refer to sections 5.2.1 and 5.3 of the JAX-WS specification for more information on resource injection.

- **Data binding with JAXB 2.2**

  JAX-WS leverages the Java Architecture for XML Binding (JAXB) 2.2 API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents. JAXB data binding replaces the data binding described by the JAX-RPC specification.

  JAX-WS 2.2 requires JAXB 2.2 for data binding. JAXB 2.2 provides minor enhancements to its annotations for improved schema generation and better integration with JAX-WS.

- **Dynamic and static clients**

  The dynamic client API for JAX-WS is called the dispatch client (javax.xml.ws.Dispatch). The dispatch client is an XML messaging oriented client. The data is sent in either `PAYLOAD` or `MESSAGE` mode. When using the `PAYLOAD` mode, the dispatch client is only responsible for providing the contents of the `<soap:Body>` and JAX-WS adds the `<soap:Envelope>` and `<soap:Header>` elements. When using the `MESSAGE` mode, the dispatch client is responsible for providing the entire SOAP envelope including the `<soap:Envelope>`, `<soap:Header>`, and `<soap:Body>` elements. JAX-WS does not add anything additional to the message. The dispatch client supports asynchronous invocations using a callback or polling mechanism.

  The static client programming model for JAX-WS is the called the proxy client. The proxy client invokes a web service based on a Service Endpoint interface (SEI), which must be provided.

- **Support for MTOM**

  Using JAX-WS, you can send binary attachments such as images or files along with web services requests. JAX-WS adds support for optimized transmission of binary data as specified by Message Transmission Optimization Mechanism (MTOM).

- **Multiple data binding technologies**

  JAX-WS exposes the following binding technologies to the end user: XML Source, SOAP Attachments API for Java (SAAJ) 1.3, and Java Architecture for XML Binding (JAXB) 2.2. XML Source enables a user to pass a `javax.xml.transform.Source` into the runtime environment which represents the data in a Source object to be processed. SAAJ 1.3 now has the ability to pass an entire SOAP document across the interface rather than just the payload itself. This action is done by the client passing the SAAJ `SOAPMessage` object across the interface. JAX-WS leverages the JAXB 2.2 support as the data binding technology of choice between Java and XML.

- **Support for SOAP 1.2**

  Support for SOAP 1.2 has been added to JAX-WS 2.0. JAX-WS supports both SOAP 1.1 and SOAP 1.2 so that you can send binary attachments such as images or files along with web services requests. JAX-WS adds support for optimized transmission of binary data as specified by MTOM.

- **Development tools**

  JAX-WS provides the **`wsgen`** and **`wsimport`** command-line tools for generating portable artifacts for JAX-WS web services. When creating JAX-WS web services, you can start with either a WSDL file or an implementation bean class. If you start with an implementation bean class, use the **`wsgen`** command-line tool to generate all the web services server artifacts, including a WSDL file if requested. If you start with a WSDL file, use the **`wsimport`** command-line tool to generate all the web services artifacts for either the server or the client. The **`wsimport`** command-line tool processes the WSDL file with schema definitions to generate the portable artifacts, which include the service class, the service endpoint interface class, and the JAXB 2.2 classes for the corresponding XML schema.

- **Support for Web Services for Java EE, version 1.3**

The Web Services for Java EE version 1.3 specification adds support for configuring the MTOM, Addressing, and RespectBinding features on JAX-WS services and clients through the use of both annotations and deployment descriptor entries.

- **Support for empty targetNamespace for the WRAPPED parameter style and return types**

  JAX-WS 2.2 supports method parameters and return types. In a JAX-WS web services operation, you can define a web services operation with an operation parameter and an optional return type. If the operation parameter and return type define an empty targetNamespace property by specifying a "" value for the targetNamespace property with either the @WebParam or @WebResult annotation, the JAX-WS runtime environment behaves in the following way:

  – If the operation is document style, the parameter style is WRAPPED, and the parameter does not map to a header, then an empty namespace is mapped with the operation parameters and return types.

  – If the parameter style is not WRAPPED, then the value of the targetNamespace parameter specified using the @WebParam or @WebResult annotation is used.

JAX-WS client programming model

The Java API for XML-Based Web Services (JAX-WS) web service client programming model supports both the Dispatch client API and the Dynamic Proxy client API. The Dispatch client API is a dynamic client programming model, whereas the static client programming model for JAX-WS is the Dynamic Proxy client. The Dispatch and Dynamic Proxy clients enable both synchronous and asynchronous invocation of JAX-WS web services.

- Dispatch client: Use this client when you want to work at the XML message level or when you want to work without any generated artifacts at the JAX-WS level.
- Dynamic Proxy client: Use this client when you want to invoke a web service based on a service endpoint interface.

Dispatch client

XML-based web services use XML messages for communications between web services and web services clients. The JAX-WS APIs provide high-level methods to simplify and hide the details of converting between Java method invocations and their associated XML messages. However, in some cases, you might desire to work at the XML message level. Support for invoking services at the XML message level is provided by the Dispatch client API. The Dispatch client API, `javax.xml.ws.Dispatch`, is a dynamic JAX-WS client programming interface. To write a Dispatch client, you must have expertise with the Dispatch client APIs, the supported object types, and knowledge of the message representations for the associated Web Services Description Language (WSDL) file. The Dispatch client can send data in either `MESSAGE` or `PAYLOAD` mode. When using the `javax.xml.ws.Service.Mode.MESSAGE` mode, the Dispatch client is responsible for providing the entire SOAP envelope including the `<soap:Envelope>`, `<soap:Header>`, and `<soap:Body>` elements. When using the `javax.xml.ws.Service.Mode.PAYLOAD` mode, the Dispatch client is only responsible for providing the contents of the `<soap:Body>` and JAX-WS includes the payload in a `<soap:Envelope>` element.

The Dispatch client API requires application clients to construct messages or payloads as XML which requires a detailed knowledge of the message or message payload. The Dispatch client supports the following types of objects:

- `javax.xml.transform.Source`: Use `Source` objects to enable clients to use XML APIs directly. You can use `Source` objects with SOAP or HTTP bindings.
- JAXB objects: Use JAXB objects so that clients can use JAXB objects that are generated from an XML schema to create and manipulate XML with JAX-WS applications. JAXB objects can only be used with SOAP or HTTP bindings.
- `javax.xml.soap.SOAPMessage`: Use `SOAPMessage` objects so that clients can work with SOAP messages. You can only use `SOAPMessage` objects with SOAP bindings.
- `javax.activation.DataSource`: Use `DataSource` objects so that clients can work with Multipurpose Internet Mail Extension (MIME) messages. Use `DataSource` only with HTTP bindings.

For example, if the input parameter type is `javax.xml.transform.Source`, the call to the Dispatch client API is similar to the following code example:

```
Dispatch<Source> dispatch = ... create a Dispatch<Source>
Source request = ... create a Source object
Source response = dispatch.invoke(request);
```

The Dispatch parameter value determines the return type of the `invoke()` method.

The Dispatch client is invoked in one of three ways:

- Synchronous invocation for requests and responses using the `invoke` method
- Asynchronous invocation for requests and responses using the `invokeAsync` method with a callback or polling object
- One-way invocation using the `invokeOneWay` methods

Refer to Chapter 4, section 3 of the JAX-WS specification for more information on using a Dispatch client.

Dynamic Proxy client

The static client programming model for JAX-WS is the called the Dynamic Proxy client. The Dynamic Proxy client invokes a web service based on a Service Endpoint Interface (SEI) which must be provided. The Dynamic Proxy client is similar to the stub client in the Java API for XML-based RPC (JAX-RPC) programming model. Although the JAX-WS Dynamic Proxy client and the JAX-RPC stub client are both based on the Service Endpoint Interface (SEI) that is generated from a WSDL file , there is a major difference. The Dynamic Proxy client is dynamically generated at run time using the Java 5 Dynamic Proxy functionality, while the JAX-RPC-based stub client is a non-portable Java file that is generated by tooling. Unlike the JAX-RPC stub clients, the Dynamic Proxy client does not require you to regenerate a stub prior to running the client on an application server for a different vendor because the generated interface does not require the specific vendor information.

The Dynamic Proxy instances extend the `java.lang.reflect.Proxy` class and leverage the Dynamic Proxy function in the base Java SE Runtime Environment (JRE) 6. The client application can then provide an interface that is used to create the proxy instance while the runtime is responsible for dynamically creating a Java object that represents the SEI.

The Dynamic Proxy client is invoked in one of three ways:

- Synchronous invocation for requests and responses using the `invoke` method
- Asynchronous invocation for requests and responses using the `invokeAsync` method with a callback or polling object
- One-way invocation using the `invokeOneWay` methods

Refer to Chapter 4 of the JAX-WS specification for more information on using Dynamic Proxy clients.

JAX-WS annotations

Java API for XML-Based Web Services (JAX-WS) relies on the use of annotations to specify metadata associated with web services implementations and to simplify the development of web services. Annotations describe how a server-side service implementation is accessed as a web service or how a client-side Java class accesses web services.

The JAX-WS programming standard introduces support for annotating Java classes with metadata that is used to define a service endpoint application as a web service and how a client can access the web service. JAX-WS supports the use of annotations based on the Metadata Facility for the Java Programming Language (Java Specification Request (JSR) 175) specification, the Web Services Metadata for the Java Platform (JSR 181) specification and annotations defined by the JAX-WS 2.0 and later (JSR 224) specification which includes JAXB annotations. Using annotations from the JSR 181 standard, you can simply annotate the service implementation class or the service interface and now the application is enabled as a web service. Using annotations within the Java source simplifies development and deployment of web services by defining some of the additional information that is typically obtained from deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL into the source artifacts.

Use annotations to configure bindings, handler chains, set names of portType, service and other WSDL parameters. Annotations are used in mapping Java to WSDL and schema, and at runtime to control how the JAX-WS runtime processes and responds to web service invocations.

For JAX-WS web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

Starting with WebSphere Application Server Version 7.0 and later, Java EE 5 application modules (Web application modules version 2.5 or above, or EJB modules version 3.0 or above) are scanned for annotations to identify JAX-WS services and clients. However, pre-Java EE 5 application modules (web application modules version 2.4 or before, or EJB modules version 2.1 or before) are not scanned for JAX-WS annotations, by default, for performance considerations.

In the Version 6.1 Feature Pack for Web Services, the default behavior is to scan pre- Java Platform, Enterprise Edition (Java EE) 5 web application modules to identify JAX-WS services and to scan pre-Java EE 5 web application modules and EJB modules for service clients during application installation. Because the default behavior for WebSphere Application Server Version 7.0 and later is to not scan pre-Java EE 5 modules for annotations during application installation or server startup, to preserve backward compatability with the feature pack from previous releases, you must configure one of the following properties:

- You can set the `UseWSFEP61ScanPolicy` property in the META-INF/MANIFEST.MF of a WAR file or EJB module to `true`. For example:

```
Manifest-Version: 1.0
UseWSFEP61ScanPolicy: true
```

  When this property is set to `true` in the META-INF/MANIFEST.MF file of the module, the module is scanned for JAX-WS annotations regardless of the Java EE version of the module. The default value is `false` and when the default value is in effect, JAX-WS annotations are only supported in modules whose version is Java EE 5 or later.
- You can set the `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` custom Java virtual machine (JVM) property using the administrative console. See the JVM custom properties documentation for the correct navigation path to use. To request annotation scanning in all modules regardless of their Java EE version, set the custom property `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` to `true`. You must change the setting on each server that requires a change in the default behavior.

If the property is set within the META-INF/MANIFEST.MF file of the module, this setting takes precedence over the server's custom JVM property. When using either property, you must establish the desired annotation scanning behavior before the application is installed. You cannot dynamically change the scanning behavior once an application is installed. If changes to the behavior are required after your application is installed, you must first uninstall the application, specify the desired scanning behavior using the appropriate property and then install the application again. When federating nodes that have the `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` set to `true` in the configuration of the servers contained within the node, this property does not affect the deployment manager. You must set the property to `true` on the deployment manager before the node is federated to preserve the behavior as it was on the node before federation.

Annotations supported by JAX-WS are listed in the table below. The target for annotations is applicable for these Java objects:

- types such as a Java class, enum or interface
- methods
- fields representing local instance variables within a Java class
- parameters within a Java method

Table 54. Web services Metadata Annotations (JSR 181). Describes the supported web services metadata annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.jws. WebService`	The **@WebService** annotation marks a Java class as implementing a Web service or marks a service endpoint interface (SEI) as implementing a web service interface. **Important:** • A Java class that implements a web service must specify either the @WebService or @WebServiceProvider annotation. Both annotations cannot be present. This annotation is applicable on a client or server SEI or a server endpoint implementation class. • If the annotation references an SEI through the `endpointInterface` attribute, the SEI must also be annotated with the @WebService annotation. • See the exposing methods in SEI-based JAX-WS web services information to learn about best practices for using the @WebService and @WebMethod annotations on a service endpoint implementation to specify Java methods that you want to expose as JAX-WS web services.	• Annotation target: Type • Properties: - **`name`** The name of the `wsdl:portType`. The default value is the unqualified name of the Java class or interface. (String) - **`targetNamespace`** Specifies the XML namespace of the WSDL and XML elements generated from the web service. The default value is the namespace mapped from the package name containing the web service. (String) - **`serviceName`** Specifies the service name of the web service: `wsdl:service`. The default value is the simple name of the Java class + *Service*. (String) - **`endpointInterface`** Specifies the qualified name of the service endpoint interface that defines the services' abstract web service contract. If specified, the service endpoint interface is used to determine the abstract WSDL contract. (String) - **`portName`** The `wsdl:portName`. The default value is WebService.name +*Port* . (String) - **`wsdlLocation`** Specifies the web address of the WSDL document defining the web service. The web address is either relative or absolute. (String)
`javax.jws. WebMethod`	The **@WebMethod** annotation denotes a method that is a web service operation. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: - **`operationName`** Specifies the name of the `wsdl:operation` matching this method. The default value is the name of Java method. (String) - **`action`** Defines the action for this operation. For SOAP bindings, this value determines the value of the SOAPAction header. The default value is the name of Java method. (String) - **`exclude`** Specifies whether to exclude a method from the web service. The default value is `false`. (Boolean)

Table 54. Web services Metadata Annotations (JSR 181) (continued). Describes the supported web services metadata annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.jws. Oneway`	The **@Oneway** annotation denotes a method as a web service one-way operation that only has an input message and no output message. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • There are no properties on the Oneway annotation.
`javax.jws. WebParam`	The **@WebParam** annotation customizes the mapping of an individual parameter to a web service message part and XML element. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Parameter • Properties: **- name** The name of the parameter. If the operation is remote procedure call (RPC) style and the `partName` attribute is not specified, then this is the name of the `wsdl:part` attribute representing the parameter. If the operation is document style or the parameter maps to a header, then `-name` is the local name of the XML element representing the parameter. This attribute is required if the operation is document style, the parameter style is `BARE`, and the mode is `OUT` or `INOUT`. (String) **- partName** Defines the name of `wsdl:part` attribute representing this parameter. This is only used if the operation is RPC style, or the operation is document style and the parameter style is `BARE`. (String) **- targetNamespace** Specifies the XML namespace of the XML element for the parameter. Applies only for document bindings when the attribute maps to an XML element. The default value is the `targetNamespace` for the web service. (String) **- mode** The value represents the direction the parameter flows for this method. Valid values are `IN`, `INOUT`, and `OUT`. (String) **- header** Specifies whether the parameter is in a message header rather than a message body. The default value is `false`. (Boolean)

Table 54. Web services Metadata Annotations (JSR 181) (continued). Describes the supported web services metadata annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.jws. WebResult`	The **@WebResult** annotation customizes the mapping of a return value to a WSDL part or XML element. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: - **`name`** Specifies the name of the return value as it is listed in the WSDL file and found in messages on the wire. For RPC bindings, this is the name of the `wsdl:part` attribute representing the return value. For document bindings, the `-name` parameter is the local name of the XML element representing the return value. The default value is `return` for RPC and DOCUMENT/WRAPPED bindings. The default value is the method name + `Response` for DOCUMENT/BARE bindings. (String) - **`targetNamespace`** Specifies the XML namespace for the return value. This parameter is only used if the operation is RPC style or if the operation is DOCUMENT style and the parameter style is BARE. (String) - **`header`** Specifies whether the result is carried in a header. The default value is `false`. (Boolean) - **`partName`** Specifies the part name for the result with RPC or DOCUMENT/BARE operations. The default value is `@WebResult.name`. (String)
`javax.jws. HandlerChain`	The **@HandlerChain** annotation associates the web service with an externally defined handler chain. You can only configure the server side handler by using the @HandlerChain annotation on the Service Endpoint Interface (SEI) or the server endpoint implementation class. Use one of several ways to configure a client side handler. You can configure a client side handler by using the @HandlerChain annotation on the generated service class or SEI. Additionally, you can programmatically register your own implementation of the HandlerResolver interface on the Service, or programmatically set the handler chain on the Binding object.	• Annotation target: Type • Properties: - **`file`** Specifies the location of the handler chain file. The file location is either an absolute java.net.URL in external form or a relative path from the class file. (String) - **`name`** Specifies the name of the handler chain in the configuration file. (String)

Table 54. Web services Metadata Annotations (JSR 181) (continued). Describes the supported web services metadata annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.jws. SOAPBinding`	The **@SOAPBinding** annotation specifies the mapping of the web service onto the SOAP message protocol. Apply this annotation to a type or methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class. The method level annotation is limited in what it can specify and is only used if the `style` property is `DOCUMENT`. If the method level annotation is not specified, the **@SOAPBinding** behavior from the type is used.	• Annotation target: Type or Method • Properties: - **`style`** Defines encoding style for messages sent to and from the web service. The valid values are `DOCUMENT` and `RPC`. The default value is `DOCUMENT`. (String) - **`use`** Defines the formatting used for messages sent to and from the web service. The default value is `LITERAL`. `ENCODED` is not supported. (String) - **`parameterStyle`** Determines whether the method's parameters represent the entire message body or whether parameters are elements wrapped inside a top-level element named after the operation. Valid values are `WRAPPED` or `BARE`. You can only use the `BARE` value with `DOCUMENT` style bindings. The default value is `WRAPPED`. (String)

Table 55. JAX-WS Annotations (JSR 224). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. Action`	The **@Action** annotation specifies the WS-Addressing action that is associated with a web service operation. When you use this annotation with a particular method, and generate the corresponding WSDL document, the WS-Addressing Action extension attribute is added to the input and output elements of the WSDL operation that corresponds to that method. To add this attribute to the WSDL operation, you must also specify the @Addressing annotation on the server endpoint implementation class. If you do not want to use the @Addressing annotation you can supply your own WSDL document with the Action attribute already defined.	• Annotation target: Method • Properties: - **`fault`** Specifies the array of `FaultAction` for the `wsdl:fault` of the operation. (String) - **`input`** Specifies the action for the`wsdl:input` of the operation. (String) - **`output`** Specifies the action for the`wsdl:output` of the operation. (String)
`javax.xml.ws. BindingType`	The **@BindingType** annotation specifies the binding to use when publishing an endpoint of this type. Apply this annotation to a server endpoint implementation class. **Important:** You can use the @BindingType annotation on the JavaBeans endpoint implementation class to enable MTOM by specifying either `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING` or `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_MTOM_BINDING` as the value for the annotation.	• Annotation target: Type • Properties: - **`value`** Indicates the binding identifier web address. Valid values are `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING`, `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING`, and `javax.xml.ws.http.HTTPBinding.HTTP2HTTP_BINDING`. The default value is `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING`. (String)
`javax.xml.ws. FaultAction`	The **@FaultAction** annotation specifies the WS-Addressing action that is added to a fault response. This annotation must be contained within an @Action annotation. When you use this annotation with a particular method, the WS-Addressing FaultAction extension attribute is added to the fault element of the WSDL operation that corresponds to that method. To add this attribute to the WSDL operation, you must also specify the @Addressing annotation on the server endpoint implementation class. If you do not want to use the @Addressing annotation you can supply your own WSDL document with the Action attribute already defined.	• Annotation target: Method • Properties: - **`value`** Specifies the action of the `wsdl:fault` of the operation. (String) - **`output`** Specifies the name of the exception class. (String) - **`className`** Specifies the name of the class representing the request wrapper. (String)

Table 55. JAX-WS Annotations (JSR 224) (continued). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. RequestWrapper`	The **@RequestWrapper** annotation supplies the JAXB generated request wrapper bean, the element name, and the namespace for serialization and deserialization with the request wrapper bean that is used at runtime. When starting with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the `className` attribute is required in this case. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: - **`localName`** Specifies the local name of the XML schema element representing the request wrapper. The default value is the `operationName` as defined in javax.jws.WebMethod annotation. (String) - **`targetNamespace`** Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String) - **`className`** Specifies the name of the class representing the request wrapper. (String) - **`partName`** Specifies the name of the wsdl:part attribute that represents the XML schema element for the RequestWrapper class. This property is applicable for JAX-WS 2.2 and later. (String)
`javax.xml.ws. ResponseWrapper`	The **@ResponseWrapper** annotation supplies the JAXB generated response wrapper bean, the element name, and the namespace for serialization and deserialization with the response wrapper bean that is used at runtime. When starting with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the `className` attribute is required in this case. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: - **`localName`** Specifies the local name of the XML schema element representing the request wrapper. The default value is the `operationName` + *Response*. `operationName` is defined in javax.jws.WebMethod annotation. (String) - **`targetNamespace`** Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String) - **`className`** Specifies the name of the class representing the response wrapper. (String) - **`partName`** Specifies the name of the wsdl:part attribute that represents the XML schema element for the ResponseWrapper class. This property is applicable for JAX-WS 2.2 and later. (String)

Table 55. JAX-WS Annotations (JSR 224) (continued). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. RespectBinding`	The **@RespectBinding** annotation specifies whether the JAX-WS implementation must use the contents of the `wsdl:binding` for an endpoint. When this annotation is specified, a check is performed to ensure all required WSDL extensibility elements with the enabled attribute set to `true` are supported. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Method • Properties: - **`enabled`** Specifies whether the `wsdl:binding` must be used or not. The default value is `true`. (Boolean)
`javax.xml.ws. ServiceMode`	The **@ServiceMode** annotation specifies whether a service provider needs to have access to an entire protocol message or just the message payload. **Important:** The @ServiceMode annotation is only supported on classes that are annotated with the @WebServiceProvider annotation.	• Annotation target: Type • Properties: - **`value`** Indicates whether the provider class accepts the payload of the message, `PAYLOAD` or the entire message `MESSAGE`. The default value is `PAYLOAD`. (String)
`javax.xml.ws. soap.Addressing`	The **@Addressing** annotation specifies that this service wants to enable WS-Addressing support. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Type • Properties: - **`enabled`** Specifies if WS-Addressing is enabled or not. The default value is `true`. (Boolean) - **`required`** Specifies that WS-Addressing headers must be present on incoming messages. The default value is `false`. (Boolean) - **`responses`** Specifies the message exchange pattern to use. The default value is `Responses.ALL`. This property is applicable for JAX-WS 2.2 and later. (String)

Table 55. JAX-WS Annotations (JSR 224) (continued). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. soap.MTOM`	The **@MTOM** annotation specifies whether binary content in the body of a SOAP message is sent using MTOM. Apply this annotation to a service endpoint implementation class.	• Annotation target: Class • Properties: - **`enabled`** Specifies if MTOM is enabled for the JAX-WS endpoint. The default value is `true`. (Boolean) - **`threshold`** Specifies the minimum size for messages that are sent using MTOM. When the message size is less than this specified integer, the message is inlined in the XML document as base64 or hexBinary data. (integer)
`javax.xml.ws. WebFault`	The **@WebFault** annotation maps WSDL faults to Java exceptions. It is used to capture the name of the fault during the serialization of the JAXB type that is generated from a global element referenced by a WSDL fault message. It can also be used to customize the mapping of service specific exceptions to WSDL faults. This annotation can only be applied to a fault implementation class on the client or server.	• Annotation target: Type • Properties: - **`name`** Specifies the local name of the XML element that represents the corresponding fault in the WSDL file. The actual value must be specified. (String) - **`targetNamespace`** Specifies the namespace of the XML element that represents the corresponding fault in the WSDL file. (String) - **`faultBean`** Specifies the name of the fault bean class. (String) - **`messageName`** Specifies the name of the wsdl:message attribute that represents the corresponding fault in the WSDL file. This property is applicable for JAX-WS 2.2 and later. (String)

Table 55. JAX-WS Annotations (JSR 224) (continued). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. WebServiceProvider`	The **@WebServiceProvider** annotation denotes that a class satisfies requirements for a JAX-WS Provider implementation class. **Important:** • A Java class that implements a web service must specify either the @WebService or @WebServiceProvider annotation. Both annotations cannot be present. • The @WebServiceProvider annotation is only supported on the service implementation class. Any class with the @WebServiceProvider annotation must implement the javax.xml.ws.Provider interface.	• Annotation target: Type • Properties: - **`targetNamespace`** Specifies the XML namespace of the WSDL and XML elements generated from the web service. The default value is the namespace mapped from the package name containing the web service. (String) - **`serviceName`** Specifies the service name of the web service: `wsdl:service`. The default value is the simple name of the Java class + *Service*. (String) - **`portName`** The `wsdl:portName`. The default value is the name of the class + *Port*. (String) - **`wsdlLocation`** The web address of the WSDL document defining the web service. This attribute is required. (String)

Table 55. JAX-WS Annotations (JSR 224) (continued). Describes the supported JAX-WS annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.xml.ws. WebServiceRef`	The **@WebServiceRef** annotation defines a reference to a web service invoked by the client. **Note:** • The @WebServiceRef annotation can be used to inject instances of JAX-WS services and ports. • The @WebServiceRef annotation is only supported in certain class types. Examples are JAX-WS endpoint implementation classes, JAX-WS handler classes, Enterprise JavaBeans classes, and servlet classes. This annotation is supported in the same class types as the @Resource annotation. See the Java Platform, Enterprise Edition (Java EE) 5 specification for a complete list of supported class types.	• Annotation target: Type, Field or Method • Properties: - **`name`** Specifies the JNDI name of the resource. The field name is the default for field annotations. The JavaBeans property name that corresponds to the method is the default for method annotations. You must specify a value for class annotations as there is no default. (String) - **`type`** Indicates the Java type of the resource. The field type is the default for field annotations. The type of the JavaBeans property is the default for method annotations. You must specify a value for class annotations as there is no default. (Class) - **`mappedName`** Specifies the name to map this resource to. (String) - **`value`** Indicates the value of the service class and it is a type that extends `javax.xml.ws.Service`. This attribute is required when the type of the reference is a service endpoint interface. (Class) - **`wsdlLocation`** The web address of the WSDL document defining the web service. This attribute is required. (String) - **`lookup`** Specifies the JNDI lookup name for the target web service. This property is applicable for JAX-WS 2.2 and later. (String)
`javax.xml.ws. WebServiceRefs`	The **@WebServiceRefs** annotation associates multiple @WebServiceRef annotations with a specific class. **Note:** The @WebServiceRef annotation is only supported in certain class types. Examples are JAX-WS endpoint implementation classes, JAX-WS handler classes, Enterprise JavaBeans classes, and servlet classes. This annotation is supported in the same class types as the @Resource annotation. See the Java Platform, Enterprise Edition (Java EE) 5 specification for a complete list of supported class types.	• Annotation target: Type • Properties: - **`value`** Specifies an array for multiple web service reference declarations. This attribute is required.

Table 56. JAX-WS Common Annotations (JSR 250). Describes the supported JAX-WS common annotations and their associated properties.

Annotation class	Annotation	Properties
`javax.annotation. Resource`	The **@Resource** annotation marks a WebServiceContext resource needed by the application. **Note:** Applying this annotation to a WebServiceContext type field on the server endpoint implementation class for a JavaBeans endpoint or a Provider endpoint results in the container injecting an instance of the WebServiceContext into the specified field. When this annotation is used in place of the @WebServiceRef annotation, the rules described for the @WebServiceRef annotation apply.	• Annotation target: Field or Method • Properties: - **`type`** Indicates the Java type of the resource. You are required to use the default, `java.lang.Object` or `javax.xml.ws.Web ServiceContext` value. If the type is the default, the resource must be injected into a field or a method. In this case, the type of the field or the type of the JavaBeans property defined by the method must be `javax.xml.ws.WebServiceContext`. (Class) If you are using this annotation to inject a web service, see the description of the @WebServiceRef `type` attribute.
`javax.annotation. Resources`	The **@Resources** annotation associates multiple @Resource annotations with a specific class and serves as a container for multiple resource declarations.	• Annotation target: Field or Method • Properties: - **`value`** Specifies an array for multiple @Resource annotations. This attribute is required.
`javax.annotation. PostConstruct`	The **@PostConstruct** annotation marks a method that needs to run after dependency injection is performed on the class. Apply this annotation to a JAX-WS application handler, a server endpoint implementation class.	• Annotation target: Method
`javax.annotation. PreDestroy`	The **@PreDestroy** annotation marks a method that must be run when the instance is in the process of being removed by the container. Apply this annotation to a JAX-WS application handler or a server endpoint implementation class.	• Annotation target: Method

Table 57. IBM proprietary annotations. Describes the supported IBM proprietary annotations and their associated properties.

Annotation class	Annotation	Properties
`com.ibm.websphere. wsaddressing. jaxws21. SubmissionAddressing`	The **@SubmissionAddressing** annotation specifies that this service wants to enable WS-Addressing support for the 2004/08 WS-Addressing specification. This annotation is part of the IBM implementation of the JAX-WS 2.1 specification. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Type • Properties: - **`enabled`** Specifies if WS-Addressing is enabled or not. The default value is `true`. (Boolean) - **`required`** Specifies that WS-Addressing headers must be present on incoming messages. The default value is `false`. (Boolean)

JAX-WS application packaging

You can package a Java Application Programming Interface (API) for XML Web Services (JAX-WS) application as a web service. A JAX-WS web service is contained within a web application archive (WAR) file or a WAR module within an enterprise archive (EAR) file.

A JAX-WS enabled WAR file contains:

- A WEB-INF/web.xml file
- Annotated classes that implement the web services contained in the application module
- [Optional] Web Services Description Language (WSDL) documents that describe the web services contained in the application module

A WEB-INF/web.xml file is similar to this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4">
</web-app>
```

The web.xml might contain servlet or servlet-mapping elements. When customizations to the web.xml file are not needed, the WebSphere Application Server runtime defines them dynamically as the module is loaded. For more information on configuring the web.xml file, see customizing web URL patterns in the web.xml file for JAX-WS applications.

Annotated classes must contain, at a minimum, a web service implementation class that includes the @WebService annotation. The definition and specification of the web services-related annotations are provided by the JAX-WS and JSR-181 specifications. The web service implementation classes can exist within the WEB-INF/classes or directory within a Java archive (JAR) file that is contained in the WEB-INF/lib directory of the WAR file.

You can optionally include WSDL documents in the JAX-WS application packaging. If the WSDL document for a particular web service is omitted, then the WebSphere Application Server runtime constructs the WSDL definition dynamically from the annotations contained in the web service implementation classes.

Starting with WebSphere Application Server Version 7.0 and later, Java EE 5 application modules (web application modules version 2.5 or above, or EJB modules version 3.0 or above) are scanned for annotations to identify JAX-WS services and clients. However, pre-Java EE 5 application modules (web application modules version 2.4 or before, or EJB modules version 2.1 or before) are not scanned for JAX-WS annotations, by default, for performance considerations. In the Version 6.1 Feature Pack for Web Services, the default behavior is to scan pre-Java EE 5 web application modules to identify JAX-WS services and to scan pre-Java EE 5 web application modules and EJB modules for service clients during application installation. Because the default behavior for WebSphere Application Server Version 7.0 and later is to not scan pre-Java EE 5 modules for annotations during application installation or server startup, to preserve backward compatability with the feature pack from previous releases, you must configure either the UseWSFEP61ScanPolicy property in the META-INF/MANIFEST.MF of a web application archive (WAR) file or EJB module or define the Java virtual machine custom property, com.ibm.websphere.webservices.UseWSFEP61ScanPolicy, on servers to request scanning during application installation and server startup. To learn more about annotations scanning, see the JAX-WS annotations information.

JAXB

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. JAXB provides the **xjc** schema compiler tool and the **schemagen** schema generator tool to transform between XML schema and Java classes.

JAXB is an XML to Java binding technology that supports transformation between schema and Java objects and between XML instance documents and Java object instances. JAXB consists of a runtime application programming interface (API) and accompanying tools that simplify access to XML documents. JAXB also helps to build XML documents that both conform and validate to the XML schema. Java API for XML-Based Web Services (JAX-WS) leverages the JAXB API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents.

This version of the application server supports the JAXB 2.2 specification. JAX-WS 2.2 requires JAXB 2.2 for data binding. JAXB 2.2 provides minor enhancements to its annotations for improved schema generation and better integration with JAX-WS.

JAXB provides the **xjc** schema compiler tool, the **schemagen** schema generator tool, and a runtime framework. You can use the **xjc** schema compiler tool to start with an XML schema definition (XSD) to create a set of JavaBeans that map to the elements and types defined in the XSD schema. You can also start with a set of JavaBeans and use the **schemagen** schema generator tool to create the XML schema. Once the mapping between XML schema and Java classes exists, XML instance documents can be converted to and from Java objects through the use of the JAXB binding runtime API. Data stored in XML documents can be accessed without the need to understand the data structure. You can then use the resulting Java classes to assemble a web services application.

JAXB annotated classes and artifacts contain all the information needed by the JAXB runtime API to process XML instance documents. The JAXB runtime API supports marshaling of JAXB objects to XML and unmarshaling the XML document back to JAXB class instances. Optionally, you can use JAXB to provide XML validation to enforce both incoming and outgoing XML documents to conform to the XML constraints defined within the XML schema.

JAXB is the default data binding technology used by the Java API for XML Web Services (JAX-WS) tooling and implementation within this product. You can develop JAXB objects for use within JAX-WS applications.

You can also use JAXB independently of JAX-WS when you want to leverage the XML data binding technology to manipulate XML within your Java applications.

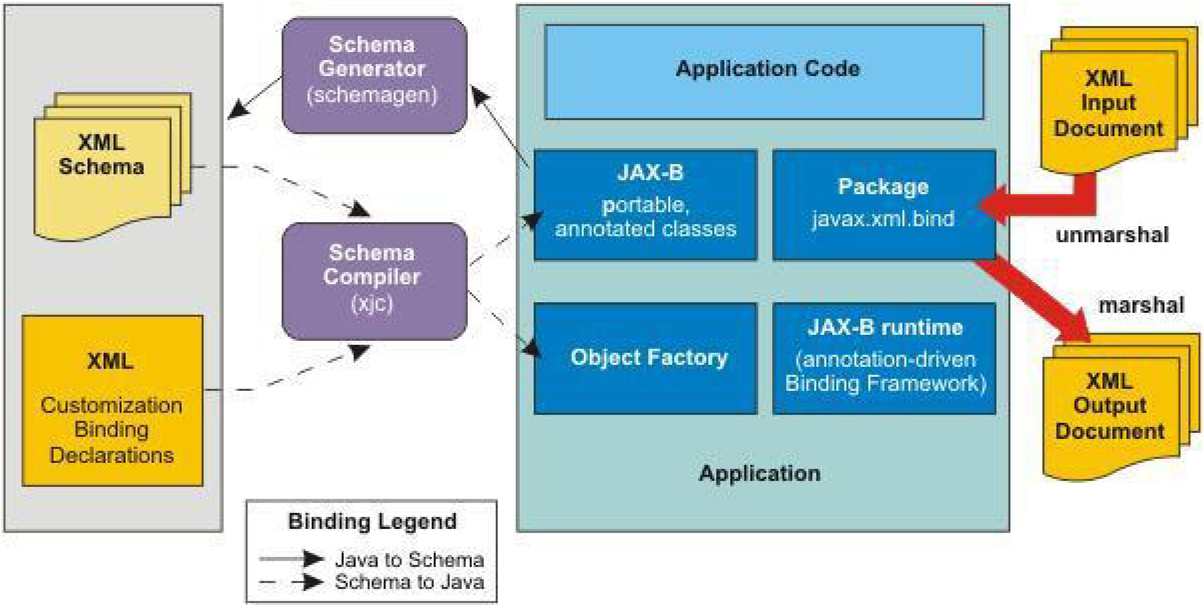


Figure 108. JAXB architecture

JAX-RPC

The Java API for XML-based RPC (JAX-RPC) specification enables you to develop SOAP-based interoperable and portable web services and web service clients. JAX-RPC 1.1 provides core APIs for developing and deploying web services on a Java platform and is a part of the Web Services for Java Platform, Enterprise Edition (Java EE) platform. The Java EE platform enables you to develop portable web services.

WebSphere Application Server implements JAX-RPC 1.1 standards.

The JAX-RPC standard covers the programming model and bindings for using Web Services Description Language (WSDL) for Web services in the Java language. JAX-RPC simplifies development of web services by shielding you from the underlying complexity of SOAP communication.

On the surface, JAX-RPC looks like another instantiation of remote method invocation (RMI). Essentially, JAX-RPC enables clients to access a web service as if the web service was a local object mapped into the client's address space even though the web service provider is located in another part of the world. The JAX-RPC is done by using the XML-based protocol SOAP, which typically rides on top of HTTP.

JAX-RPC defines the mappings between the WSDL port types and the Java interfaces, as well as Java language and Extensible Markup Language (XML) schema types.

A JAX-RPC web service can be created from a JavaBeans or a enterprise bean implementation. You can specify the remote procedures by defining remote methods in a Java interface. You only need to code one or more classes that implement the methods. The remaining classes and other artifacts are generated by the web service vendor's tools. The following is an example of a web service interface:

```
package com.ibm.mybank.ejb;
import java.rmi.RemoteException;
import com.ibm.mybank.exception.InsufficientFundsException;
/**
 * Remote interface for Enterprise Bean: Transfer
 */
public interface Transfer_SEI extends java.rmi.Remote {
   public void transferFunds(int fromAcctId, int toAcctId, float amount)
      throws java.rmi.RemoteException;

}
```

The interface definition in JAX-RPC must follow specific rules:

- The interface must extend `java.rmi.Remote` just like RMI.
- Methods must create `java.rmi.RemoteException`.
- Method parameters cannot be remote references.
- Method parameter must be one of the parameters supported by the JAX-RPC specification. The following list are examples of method parameters that are supported. For a complete list of method parameters see the JAX-RPC specification.
  - Primitive types: `boolean`, `byte`, `double`, `float`, `short`, `int` and `long`
  - Object wrappers of primitive types: `java.lang.Boolean`, `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`
  - `java.lang.String`
  - `java.lang.BigDecimal`
  - `java.lang.BigInteger`
  - `java.lang.Calendar`
  - `java.lang.Date`
- Methods can take value objects which consist of a composite of the types previously listed, in addition to aggregate value objects.

A client creates a stub and invokes methods on it. The stub acts like a proxy for the web service. From the client code perspective, it seems like a local method invocation. However, each method invocation gets marshaled to the remote server. Marshaling includes encoding the method invocation in XML as prescribed by the SOAP protocol.

The following are key classes and interfaces needed to write web services and web service clients:

- Service interface: A factory for stubs or dynamic invocation and proxy objects used to invoke methods
- ServiceFactory class: A factory for Services.
- `loadService`

  The `loadService` method is provided in WebSphere Application Server Version 6.0 to generate the service locator which is required by a JAX-RPC implementation. If you recall, in previous versions there was no specific way to acquire a generated service locator. For managed clients you used a JNDI method to get the service locator and for non-managed clients, you were required to instantiate IBM's specific service locator `ServiceLocator service=new ServiceLocator(...);` which does not offer portability. The `loadService` parameters include:
  - `wsdlDocumentLocation`: A URL for the WSDL document location for the service or null.
  - `serviceName`: A qualified name for the service
  - `properties`: A set of implementation-specific properties to help locate the generated service implementation class.
- `isUserInRole`

  The `isUserInRole` method returns a boolean indicating whether the authenticated user for the current method invocation on the endpoint instance is included in the specified logical role.
  - `role`: The role parameter is a String specifying the name of the role.
- Service
- Call interface: Used for dynamic invocation
- Stub interface: Base interface for stubs

If you are using a stub to access the web service provider, most of the JAX-RPC API details are hidden from you. The client creates a ServiceFactory (`java.xml.rpc.ServiceFactory`). The client instantiates a Service (`java.xml.rpc.Service`) from the ServiceFactory. The service is a factory object that creates the port. The port is the remote service endpoint interface to the web service. In the case of DII, the Service object is used to create Call objects, which you can configure to call methods on the Web service's port.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

RMI-IIOP using JAX-RPC

Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) can be used with JAX-RPC to support non-SOAP bindings.

Java API for XML-based Remote Procedure Call (JAX-RPC) is the Java standard API for invoking web services through remote procedure calls. A transport is used by a programming language to communicate over the Internet. You can use protocols with the transport such as SOAP and Remote Method Invocation (RMI). You can use Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) with JAX-RPC to support non-SOAP bindings.

Using RMI-IIOP with JAX-RPC, enables WebSphere Java clients to invoke enterprise beans using a WSDL file and the JAX-RPC programming model instead of using the standard Web Services for Java Platform, Enterprise Edition (Java EE) programming model. When a enterprise JavaBeans implementation is used to invoke a web service, multiprotocol JAX-RPC permits the web service invocation path to be optimized for WebSphere Java clients. To learn more this optimization, read about using enterprise bean bindings to invoke an EJB from a web services client.

Benefits of using the RMI/IIOP protocol instead of a SOAP- based protocol are:

- XML processing is not required to send and receive messages; Java serialization is used instead.
- The client JAX-RPC call can participate in a user transaction, which is not the case when SOAP is used.

WS-I Basic Profile

The Web Services-Interoperability (WS-I) Basic Profile is a set of non-proprietary web services specifications that promote interoperability. WebSphere Application Server conforms to the WS-I Basic Profile Version 1.1 and WS-I Basic Security Profile Version 1.0.

The WS-I Basic Profile is governed by a consortium of industry-leading corporations, including IBM, under direction of the WS-I Organization. The profile consists of a set of principles that relate to bringing about open standards for web services technology. All organizations that are interested in promoting interoperability among web services are encouraged to become members of the Web Services Interoperability Organization.

Several technology components are used in the composition and implementation of web services, including messaging, description, discovery, and security. Each of these components are supported by specifications and standards, including SOAP 1.1, Extensible Markup Language (XML) 1.0, HTTP 1.1, Web Services Description Language (WSDL) 1.1, and Universal Description, Discovery and Integration (UDDI). The WS-I Basic Profile specifies how these technology components are used together to achieve interoperability, and mandates specific use of each of the technologies when appropriate. You can read more about the WS-I Basic Profile at the WS-I Organization website.

As technology components are updated, these components are also used in the composition and implementation of web services. One example is that both SOAP 1.1 and SOAP 1.2 are now supported.

Building on the support for WS-I Basic Profile Version 1.0, WS-I Basic Profile V1.1, Attachment Profile V1.0, Basic Security Profile (BSP) V1.0, and WS-I Basic Security Profile V1.1, you can implement web services with this product using the following active WS-I profiles:

- *WS-I Basic Profile V1.2* builds on WS-I Basic Profile V1.0 and WS-I Basic Profile V1.1 and adds support for WS-Addressing (WS-A) and SOAP Message Transmission Optimization Mechanism (MTOM). The WS-Addressing specification enables the asynchronous message exchange pattern so that you can decouple the service request from the service response. The SOAP header of the sender's request contains the `wsa:ReplyTo` value that defines the endpoint reference to which the provider's response is sent. Decoupling the request from the response enables long running web services interactions. Leveraging the asynchronous programming model support in JAX-WS Version 2.1 in combination with WS-Addressing, you can now take advantage of the ability to create web services invocations where the client can continue to process work without waiting for a response to return. This provides for a more dynamic and efficient model to invoke web services. Using MTOM, you can send and receive binary data optimally within a SOAP message.

- *WS-I Basic Profile V2.0* builds on Basic Profile V1.2 with the addition of support for SOAP 1.2.
- *WS-I Reliable Secure Profile 1.0* builds on WS-I Basic Profile V1.2, WS-I Basic Profile V2.0, WS-I Basic Security Profile V1.0, and WS-I Basic Security Profile V1.1 and adds support for WS-Reliable Messaging 1.1, WS-Make Connection 1.0, and WS-Secure Conversation 1.3. WS-Reliable Messaging 1.1 is a session-based protocol that provides message level reliability for web services interactions. WS-Make Connection 1.0 was developed by the WS-Reliable Messaging workgroup to address scenarios where a web services endpoint is behind a firewall or the endpoint has no visible endpoint reference. If a web services endpoint loses connectivity during a reliable session, WS-Make Connection provides an efficient method to re-establish the reliable session. Additionally, WS-Secure Conversation V1.3 is a session-based security protocol that uses an efficient symmetric key based encryption algorithm for message level security. WS-I Reliable Secure Profile V1.0 provides secure reliable session-oriented web services interactions.

Each of the technology components has requirements that you can read about in more detail at the WS-I Organization website. For example, support for Universal Transformation Format (UTF)-16 encoding is required by WS-I Basic Profile. UTF-16 is a kind of Unicode encoding scheme that uses 16-bit values to store Universal Character Set (UCS) characters. UTF-8 is the most common encoding that is used on the Internet; UTF-16 encoding is typically used for Java and Windows product applications; and UTF-32 is used by various Linux and UNIX systems. Unlike UTF-8, UTF-16 has issues with big-endian and little-endian, and often involves Byte Order Mark (BOM) to indicate the endian. BOM is mandatory for UTF-16 encoding and it can be used in UTF-8.

The application server only supports UTF-8 and UTF-16 encoding of SOAP messages.

See the information on changing SOAP message encoding to support WSI-Basic Profile to learn how to modify your encoding from UTF-8 to UTF-16.

Table 58. UTF properties. Specifies the corresponding bytes and encoding form for UTF properties.

Bytes	Encoding form
EF BB BF	UTF-8
FF FE	UTF-16, little-endian
FE FF	UTF-16, big-endian
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian

BOM is written prior to the XML text, and it indicates to the parser how the XML is encoded. The XML declaration contains the encoding, for example: `<?xml version=xxx encoding="utf-xxx"?>`. BOM is used with the encoding to determine how to interpret the XML. Here is an example of a SOAP message and how BOM and UTF encoding are used:

```
POST http://www.whitemesa.net/soap12/add-test-rpc HTTP/1.1
Content-Type: application/soap+xml; charset=utf-16; action=""
SOAPAction:
Host: localhost: 8080
Content-Length: 562

OxFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
      xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
      xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
      xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <q1:echoString xmlns:q1="http://soapinterop.org/">
      <inputString soap:encodingStyle="http://example.org/unknownEncoding"
```

```
                  xsi:type="xsd:string">
        Hello SOAP 1.2
      </inputString>
    </q1:echoString>
  </soap:Body>
</soap:Envelope>
```

In the example code, 0xFF0xFE represents the byte codes, while the *<?xml>* declaration is the textual representation.

Support for styleEncoding is not supported in SOAP 1.2 so here is the same example of the SOAP message but without the encoding information:

```
0xFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
      xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
      xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
      xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <q1:echoString xmlns:q1="http://soapinterop.org/">
      <inputString xsi:type="xsd:string">
        Hello SOAP 1.2
      </inputString>
    </q1:echoString>
  </soap:Body>
</soap:Envelope>
```

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

WS-I Attachments Profile

The *Web Services-Interoperability (WS-I) Attachments Profile* is a set of non-proprietary web services specifications that promote interoperability. This profile compliments the WS-I Basic Profile 1.1 to add support for interoperable SOAP messages with attachments-based web services.

WebSphere Application Server conforms to the WS-I Attachments Profile 1.0.

Attachments are typically used to send binary data, for example, data that is mapped in Java code to java.awt.Image and javax.activation.DataHandler. The raw data can be sent in the SOAP message, however, this approach is inefficient because an XML parser has to scan the data as it parses the message.

The WS-I Attachments Profile provides a solution to the limitations that are presented by Web Services Description Language (WSDL) 1.1. Because WSDL 1.1 attachments are not part of the XML schema type space, they can be message parts only. As message parts, the attachments cannot be arrays or properties of Java beans. The profile defines the wsi:swaRef XML schema type.Use the wsi:swaRef XML schema type to overcome the limitations of WSDL 1.1 attachments.

The wsi:swaRef type is an extension of the xsd:anyURI type, where its value contains the content-ID of the attachment.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Overview of IBM JAX-RS

Java API for RESTful Web Services (JAX-RS) is a programming model that provides a mechanism for developing services that follow Representational State Transfer (REST) principles. Using JAX-RS, development of RESTful services is simplified.

JAX-RS is a collection of interfaces and Java annotations that simplifies development of server-side REST applications. By using JAX-RS technology, REST applications are simpler to develop, simpler to consume, and simpler to scale when compared to other types of distributed systems. This product supports a Java API for developing REST-based services. The IBM implementation of JAX-RS provides an implementation of the JAX-RS specification.

To develop RESTful services using IBM JAX-RS, it is assumed that you are familiar with basic REST principles and a basic knowledge of standard technologies, such as HTTP, and XML.

REST and JAX-RS

Representational State Transfer, also known as REST, is an architectural style that uses multiple standard technologies like HTTP, XML, ATOM, and HTML. REST is used to define flexible applications based on the notion of resources. A resource is simply any data that you want to share on the web that you can identify by a Uniform Resource Identifier (URI).

JAX-RS is a specification defined by JSR-311 in the Java Community Process. Some of the key features provided by JAX-RS include:

- A collection of annotations for declaring resource classes and the data types they support
- A set of interfaces that allow application developers to gain access to the runtime context
- An extensible framework for integrating custom content handlers

Apache Wink and the IBM implementation of JAX-RS

Wink is a project developed within the Apache Software Foundation that provides a lightweight framework for developing RESTful applications. Wink supports REST services implemented using JAX-RS to describe the resources on the server. However, a client API is also provided by Wink. This client API is specific to the Wink runtime environment because there is no JAX-RS defined client API.

The IBM implementation of JAX-RS is an extension of the base Wink 1.1 runtime environment. IBM JAX-RS includes the following features:

- JAX-RS 1.1 server runtime
- Stand-alone client API with the option to use Apache HttpClient 4.0 as the underlying client
- Built-in entity provider support for JSON4J
- An Atom JAXB model in addition to Apache Abdera support
- Multipart content support
- A handler system to integrate user handlers into the processing of requests and responses

Now, you are ready to start learning more about implementing RESTful services using IBM JAX-RS.

- For an example of how to get a JAX-RS web application running quickly, see the quick start documentation.
- To learn about planning considerations for the JAX-RS application, see the planning to use JAX-RS to enable RESTful services documentation.
- To learn more about developing, packaging, and deploying JAX-RS web services, see the implementing JAX-RS web applications documentation. Additional information is provided for implementing JAX-RS web applications that use XML, JSON, or Atom content formats.

Web Services Addressing support

The Web Services Addressing (WS-Addressing) support in this product provides the environment for web services that use the World Wide Web Consortium (W3C) WS-Addressing specifications. This family of specifications provide transport-neutral mechanisms to address web services and to facilitate end-to-end addressing.

You do not usually have to be aware of the underlying WS-Addressing support because WebSphere Application Server ensures that your web service applications are WS-Addressing compliant when required. Read this topic only if you have to use the WS-Addressing support directly. For example, if you have one of the following roles:

- A web service developer who needs to use the WS-Addressing application programming interfaces (APIs) to create endpoint references within an application, and then use these references to target web service resource instances.
- A system programmer who needs to use the IBM proprietary WS-Addressing system programming interfaces (SPIs) to undertake more advanced WS-Addressing operations, such as specifying message-addressing properties on web services messages.
- An administrator who is configuring policy sets for JAX-WS applications.

The WS-Addressing support for developers consists of two sets of programming interfaces: the JAX-WS standard interfaces, and the IBM proprietary implementation of the WS-Addressing specification.

Features of the JAX-WS WS-Addressing support

This product provides support for the JAX-WS WS-Addressing APIs, which you can use to undertake basic addressing functions such as creating an endpoint reference, enabling WS-Addressing support, and specifying the action URIs that are associated with the WSDL operations of the web service. Use these APIs if you want to undertake simple WS-Addressing functions and create JAX-WS applications that are portable.

The JAX-WS WS-Addressing APIs provide the following features for core WS-Addressing application development:

- Java representations of WS-Addressing endpoint references.
  - You can create Java endpoint reference instances for the application endpoint, or other endpoints in the same application, at run time. You do not have to specify the URI of the endpoint reference.
  - You can create Java endpoint reference instances for endpoints in other applications by specifying the URI of the endpoint reference.
  - On services, you can use annotations to specify whether WS-Addressing support is enabled, whether it is required, and which message exchange pattern (synchronous, asynchronous, or both) to use.
  - On clients, you can use features to specify whether WS-Addressing support is enabled and whether it is required.
  - You can configure client proxy or Dispatch objects by using endpoint references.
- Java support for endpoint references that represent Web Services Resource (WS-Resource) instances.
  - You can associate reference parameters with an endpoint reference at the time of its creation, to correlate it with a particular resource instance.
  - In targeted web services, you can extract the reference parameters of an incoming message, so that the web service can route the message to the appropriate WS-Resource instance.

The following features were introduced in the JAX-WS 2.2 specification, which WebSphere Application Server supports from Version 8:

- You can specify additional binding information within the metadata of an endpoint reference as part of the JAX-WS 2.2 specification. This functionality was added to WebSphere Application Server in Version 7, however as it was not part of the JAX-WS 2.1 specification, you might have experienced incompatibility issues when interoperating with non-WebSphere Application Server servers which did not offer support for additional metadata in endpoint references. JAX-WS 2.2 supports WSDL metadata in endpoint references, so applications will be compatible with other implementations of this specification. See the “Web Services Addressing overview”topic for further information.
- You can enable and configure WS-Addressing on a client or service by adding WS-Policy assertions into the WSDL document. WebSphere Application Server processes WS-Addressing information held within the WS-Policy aspect of a WSDL document and uses it in the configuration of that application. See the “Enabling Web Services Addressing support for JAX-WS applications using WS-Policy” topic for further information.

- You can specify whether a synchronous or an asynchronous message exchange pattern is required by a web service application using the addressing annotation or the AddressingFeature. Use the `responses` parameter on the addressing annotations or the AddressingFeature class in the code. See the “Enabling Web Services Addressing support for JAX-WS applications using addressing annotations” topic and the “Enabling Web Services Addressing support for JAX-WS applications using addressing features” topic for further information.
- You can configure WS-Addressing using deployment descriptors. Add an `<addressing>` element and optional child elements to the deployment descriptor file for the application. See the “Enabling Web Services Addressing support for JAX-WS applications using deployment descriptors” topic for further information.
- You can generate code from a WSDL document and WebSphere Application Server automatically inserts @Action and @FaultAction annotations into the generated Java code. See the “Web Services Addressing annotations” topic for further information.

Features of the IBM proprietary WS-Addressing support

This product provides an IBM proprietary implementation of the WS-Addressing specification, which you can use with JAX-RPC applications as well as JAX-WS applications, to undertake more advanced functions such as creating endpoint references that represent highly available objects, or directly setting message addressing properties in the SOAP header. Use these APIs and SPIs if you want to create JAX-RPC applications that use addressing, or you want to undertake more advanced functions that are not possible with the JAX-WS APIs.

The IBM proprietary API provides the following basic features:

- You can easily create Java endpoint reference instances to represent any endpoint in the server, based on the deployment environment of the application. You do not have to specify the URI of the endpoint reference. Additionally, endpoint references can represent highly available or workload-managed objects.
- You can configure client JAX-WS BindingProvider request context objects, or JAX-RPC Stub or Call objects, with a WS-Addressing endpoint reference. Future invocations on these objects are targeted at the endpoint that is represented by the endpoint reference. The invocations also automatically conform to the WS-Addressing specification (namespace) that is associated with that endpoint reference.

The IBM proprietary WS-Addressing SPIs provide support for extended WS-Addressing system development by using the following features:

- Reasoning and manipulation of endpoint references beyond what is available at the application programming level.
  - You can manipulate the contents of the endpoint reference, as specified by the WS-Addressing specification.
  - You can associate a WS-Addressing namespace, and therefore specification behavior, with an endpoint reference.
- Java representations of the WS-Addressing message-addressing properties.
  - You can specify WS-Addressing message-addressing properties for outbound web service messages. In the targeted web service, you can extract message addressing properties from inbound web service messages.
  - You can specify the WS-Addressing namespace of an outbound WS-Addressing message, although in most cases the namespace is automatically derived based on the target endpoint reference. In a targeted web service, you can acquire the WS-Addressing namespace of an incoming message.

Support for WS-Addressing specifications and interoperability

By default, this product supports the W3C WS-Addressing 1.0 Core and SOAP Binding specifications that are identified by the http://www.w3.org/2005/08/addressing namespace. Unless otherwise stated, WS-Addressing semantics that are described in this documentation refer to these specifications.

For interoperability, other levels of the WS-Addressing specification are supported in this version of the product; in particular, the WS-Addressing W3C submission with the namespace http://schemas.xmlsoap.org/ws/2004/08/addressing.

For JAX-WS applications, this product supports the WS-Addressing metadata specification identified by the http://www.w3.org/2007/05/addressing/metadata namespace. This specification supersedes the WS-Addressing Web Services Description Language (WSDL) binding specification identified by the http://www.w3.org/2006/05/addressing/wsdl namespace.

In addition, this product supports the following features from the WS-Addressing WSDL binding specification:

- The wsaw:UsingAddressing extensibility element, on the WSDL Binding element only. The supported namespaces for this element are the http://www.w3.org/2006/05/addressing/wsdl namespace and the http://www.w3.org/2006/02/addressing/wsdl namespace (deprecated).
- The wsaw:Action extensibility element. The supported namespaces for this element are the http://www.w3.org/2006/05/addressing/wsdl namespace, the http://www.w3.org/2006/02/addressing/wsdl namespace (deprecated), and the http://schemas.xmlsoap.org/ws/2004/08/addressing namespace.

Web Services Addressing overview

Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between web services by defining a standard way to address web services and provide addressing information in messages. The WS-Addressing specification introduces two primary concepts: endpoint references, and message-addressing properties. For further details, refer to the WS-Addressing specifications.

Endpoint references

Endpoint references provide a standard mechanism to encapsulate information about specific endpoints. Endpoint references can be propagated to other parties and then used to target the web service endpoint that they represent. The following table summarizes the information model for endpoint references.

Table 59. Information model for endpoint references. The table lists the different abstract property names and for each one shows their property type, multiplicity and brief description.

Abstract property name, using the notational convention of the W3C XML Information Set	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters]*	xs:any	0..unbounded	Namespace qualified element information items that are required to interact with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies and capabilities of the endpoint.

The following prefix and corresponding namespace is used in the previous table.

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema

The following XML fragment illustrates an endpoint reference. This element references the endpoint at the URI http://example.com/fabrikam/acct, has metadata specifying the interface to which the endpoint reference refers, and has application-defined reference parameters of the http://example.com/fabrikam namespace.

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                       xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
                       xmlns:fabrikam="http://example.com/fabrikam"
                       xmlns:wsdli="http://www.w3.org/2005/08/wsdl-instance"
                       wsdli:wsdlLocation="http://example.com/fabrikam
                       http://example.com/fabrikam/fabrikam.wsdl">
  <wsa:Address>http://example.com/fabrikam/acct</wsa:Address>
  <wsa:Metadata>
    <wsam:InterfaceName>fabrikam:Inventory</wsam:InterfaceName>
  </wsa:Metadata>
  <wsa:ReferenceParameters>
    <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
    <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
  </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Message-addressing properties

Message addressing properties (MAPs) are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers and provide a standard way of conveying information, such as the endpoint to which message replies should be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Table 60. Message-addressing properties defined by the WS-Addressing specification. The table lists the abstract WS-Addressing MAP names and for each one shows their MAP content type, multiplicity and brief description.

Abstract WS-Addressing MAP name, using the notational convention of the W3C XML Information Set	MAP content type	Multiplicity	Description
[action]	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This proprety corresponds to the address property of the endpoint reference to which the message is addressed. This value is required.
[destination]	xs:anyURI	1..1	The absolute URI that specifies the address of the intended receiver of this message. This value is optional because, if not present, it defaults to the anonymous URI that is defined in the specification, indicating that the address is defined by the underpinning protocol.
[reference parameters]*	xs:any	0..unbounded	Correspond to the reference parameters property of the endpoint reference to which the message is addressed. This value is optional.
[source endpoint]	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional.
[fault endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional.
[relationship]*	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The content of this element conveys the message ID of the related message. An optional attribute conveys the relationship type. This value is optional.
[message id]	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional.

The abstract names in the previous tables are used to refer to the MAPs throughout this documentation.

The following example of a SOAP message contains WS-Addressing MAPs:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
           xmlns:wsa="http://www.w3.org/2005/08/addressing"
           xmlns:fabrikam="http://example.com/fabrikam">
  <S:Header>
    ...
    <wsa:To>http://example.com/fabrikam/acct</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address> http://example.com/fabrikam/acct</wsa:address>
    </wsa:ReplyTo>
    <wsa:Action>...</wsa:Action>
    <fabrikam:CustomerKey wsa:IsReferenceParameter='true'>123456789</fabrikam:CustomerKey>
    <fabrikam:ShoppingCart wsa:IsReferenceParameter='true'>ABCDEFG</fabrikam:ShoppingCart>
    ...
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Web Services Addressing message exchange patterns

The World Wide Web Consortium (W3C) Web Services Addressing (WS-Addressing) specification explicitly defines the WS-Addressing core properties for the message exchange patterns (MEPs) that are defined by WSDL 1.0. These MEPs are summarized in this topic, illustrating the mandatory WS-Addressing properties for each pattern.

One-way MEP

This straightforward one-way message is defined in WSDL 1.0 as an input-only operation. The WSDL fragment for this operation has the following form:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
</operation>
```

The following WS-Addressing message addressing properties (MAPs) are automatically added to the message header of a one-way WS-Addressing input message by the client application server run time, to ensure compliance with the WS-Addressing specification.

Tip: You can override these values by using the IBM proprietary WS-Addressing system programming interfaces (SPIs).

Table 61. The WS-Addressing message addressing properties that a client adds to the message header of a one-way WS-Addressing input message. The table lists the different WS-Addressing MAP names and provides a description for each one.

Abstract WS-Addressing MAP name, using the notational convention of the W3C XML Information Set	Description for a one-way input message
[action]	The WS-Addressing action that is generated in accordance with the version of the WS-Addressing specification that is in use.
[reply endpoint]	The WS-Addressing reply endpoint indicating that no replies are expected to this input message. The value of this MAP depends on the version of the WS-Addressing specification that is in use.
[message id]	A uniquely generated message identifier. Although not mandated by the specification, the WebSphere Application Server run time automatically sets this value.

Although the WSDL operation for this message exchange does not specify any responses, related messages can be sent as part of other message exchanges. In particular, applications can use the WS-Addressing reply endpoint or fault endpoint MAPs to indicate to the target of a one-way message where to send related messages. To propagate a reply endpoint or fault endpoint, associate the appropriate property with the request context for the JAX-WS BindingProvider object, or with the JAX-RPC Stub or Call object, as described in Specifying and acquiring message-addressing properties by using the IBM proprietary Web Services Addressing SPIs, to override the defaults.

Two-way request-response

This is a request-response MEP as defined in WSDL 1.1. The response part of the operation might be defined as an output message, or a fault message, or both. The following WSDL code extracts show the various forms of definition for such an operation:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <output message="tns:myOutputMessage"/>
  <fault="tns:myFaultMessage"/>
</operation>
```

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <output message="tns:myOutputMessage"/>
</operation>
```

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <fault="tns:myFaultMessage"/>
</operation>
```

The application server client run time ensures that the SOAP header of the outgoing request message contains the relevant WS-Addressing message information headers. The calling application does not have to set the WS-Addressing headers. A response is expected, therefore you must specify a reply endpoint or fault endpoint in the request message.

Tip: In the 2005/08 specification, an unspecified reply endpoint is valid because it defaults to an endpoint reference that contains the anonymous URI.

The following table summarizes the MAPs that the product sets by default on a web service request that uses WS-Addressing protocol. You can override or specify other MAPs by using the IBM proprietary WS-Addressing SPIs.

Table 62. The message addressing properties that are added on a web service request that uses the WS-Addressing protocol. The table lists the different WS-Addressing MAP names and provides a description for each one.

Abstract WS-Addressing MAP name, using the notational convention of the W3C XML Information Set	Description for the input message of a request-response operation
[action]	The WS-Addressing action that is generated in accordance with the version of the WS-Addressing specification that is in use.
[message id]	A uniquely generated message identifier.

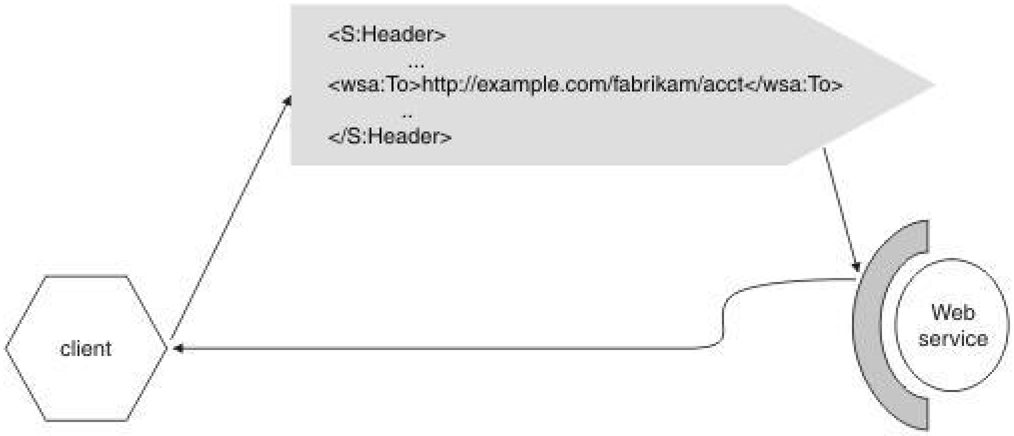
The following table summarizes the MAPs that are set by default by the product on a WS-Addressing response or fault message.

Table 63. The message addressing properties that are added on a WS-Addressing response or fault message. The table lists the different WS-Addressing MAP names and provides a description for each one.

Abstract WS-Addressing MAP name, using the notational convention of the W3C XML Information Set	Description for the input message of a request-response operation
[action]	The WS-Addressing action that is generated in accordance with the version of the WS-Addressing specification that is in use.
[relationship]	A relationship set containing a reply relationship to the message id that is passed in the request message.
[message id]	A uniquely generated message identifier; although not mandated by the specification, the application server run time automatically sets this property.

Synchronous request-response

By default, if you do not use the IBM proprietary WS-Addressing SPI to set the reply endpoint or fault endpoint, the response part of a two-way message is returned according to the underlying protocol in use. In particular, for an HTTP request, the response is returned synchronously in the HTTP response.



For JAX-WS synchronous invocations, if you set the reply endpoint or the fault endpoint, the endpoint reference that you set must use the anonymous URI. If the endpoint reference does not use the anonymous URI, a javax.xml.ws.WebServiceException exception is thrown. Although the endpoint reference uses the anonymous URI, you can use reference parameters within the endpoint reference to target the reply or fault endpoint.

For JAX-WS applications, you can specify a synchronous message exchange pattern by applying and configuring a WS-Addressing policy type. This exchange pattern is particularly useful in the following scenarios:

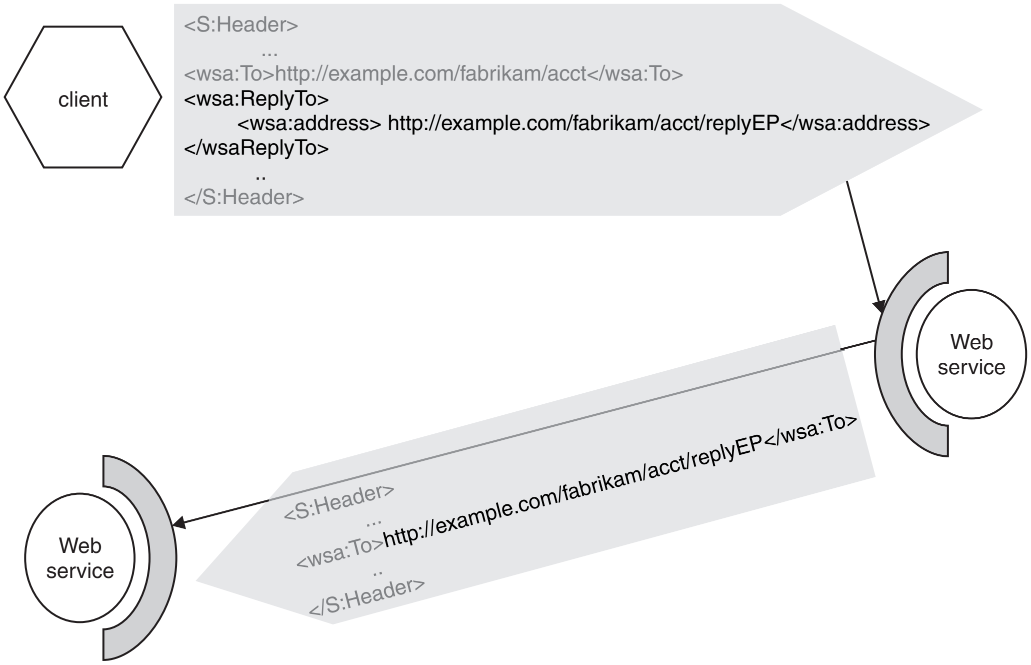
- You do not have WS-Security enabled, or have not used an assembly tool to specify that the ReplyTo and FaultTo elements of the SOAP message should be signed. In this situation, it is possible for a JAX-WS endpoint to be used to send messages to a third party, potentially taking part in a Denial of Service attack. To prevent such attacks, specify the synchronous message exchange pattern, and enable WS-Policy so that clients are aware of this requirement.
- A JAX-WS client is communicating through a NAT device. URIs in the ReplyTo or FaultTo elements of the SOAP message cannot be routed through such a device. In this situation, the client must use the anonymous URI defined by the WS-Addressing specification, and a synchronous message exchange pattern. To ensure that the client conforms to these requirements even if the server uses WS-Policy to request a non-anonymous URI in the ReplyTo element, specify the synchronous message exchange pattern on the client.

You can ensure that servers or clients are aware of the requirement for synchronous messaging by enabling WS-Policy.

Asynchronous request-response

The JAX-RPC 1.0 programming model does not allow for asynchronous replies or faults to a two-way request-response operation.

Responses to, or faults generated from, requests that are directed at endpoints hosted on WebSphere Application Server are targeted at the reply endpoint or fault endpoint, in accordance with the WS-Addressing specification. The connection with the requesting client will be closed with an HTTP 202 response.


client
<S:Header>
...
<wsa:To>http://example.com/fabrikam/acct</wsa:To>
<wsa:ReplyTo>
<wsa:address> http://example.com/fabrikam/acct/replyEP</wsa:address>
</wsaReplyTo>
..
</S:Header>
Web service
<S:Header>
...
<wsa:To>http://example.com/fabrikam/acct/replyEP</wsa:To>
..
</S:Header>
Web service

For JAX-WS asynchronous invocations, the reply endpoint is generated automatically for use by the JAX-WS implementation. If you attempt to set either a reply endpoint or a fault endpoint, a javax.xml.ws.WebServiceException exception is thrown.

For JAX-WS applications, you can specify an asynchronous message exchange pattern in several different ways.

- By applying and configuring a WS-Addressing policy set. See the Configuring the WS-Addressing policy topic.
- By setting the `com.ibm.websphere.webservices.use.async.mep` property on the client request context. See the Invoking JAX-WS web services asynchronously topic.
- Through the use of deployment descriptor elements, @Addressing annotations, addressing features, or by adding WS-Policy assertions into the WSDL document. See the Enabling Web Services Addressing support for JAX-WS applications topic and its child topics.

This exchange pattern is particularly useful if a JAX-WS endpoint has a long-running invocation time. Client and server resources are used to keep the connection open, but this use of resource might be impractical if the service takes a long time to respond.

The message exchange pattern configuration is expressed in WS-Policy attachments in the WSDL document. Clients will be able to access this message exchange pattern configuration information if any of the following conditions are true:

- WS-Policy sharing is enabled.
- WS-Policy sharing is not enabled, but:
  - the packaged WSDL (as retrived by an HTTP GET request) contains the relevant policy information.
  - @Addressing annotations have been used in the code. In this case, the server runtime generates a WSDL document containing the WS-Policy attachments.

See the “Web service providers and policy configuration sharing” on page 829 topic for further details.

Web Services Addressing version interoperability

The Web Services Addressing (WS-Addressing) support in this product can interoperate with various versions of the WS-Addressing specification.

Table 64. Supported set of WS-Addressing versions. The table lists the associated namespace, the specification download locations and some details about each specification.

Associated namespace	Specification download location	Details
http://www.w3.org/2005/08/addressing	http://www.w3.org/2002/ws/addr/	W3C final versions of the WS-Addressing core and SOAP specifications. These specifications are sometimes referred to collectively as the 2005/08 version of WS-Addressing.
http://www.w3.org/2007/05/addressing/metadata	http://www.w3.org/2002/ws/addr/	W3C final version of the WS-Addressing metadata specification. This specification defines WS-Addressing WSDL extensions and WS-Policy assertions. For JAX-WS applications, this specification supersedes the http://www.w3.org/2006/05/addressing/wsdl specification.
http://www.w3.org/2006/05/addressing/wsdl	http://www.w3.org/2002/ws/addr/	W3C Candidate Recommendation (CR) version of the WS-Addressing WSDL specification. This is the default namespace used by this product for the WSDL parts of the WS-Addressing specification, for JAX-RPC applications. For JAX-WS applications, this specification is superseded by the http://www.w3.org/2007/05/addressing/metadata specification.
http://www.w3.org/2006/02/addressing/wsdl	http://www.w3.org/2002/ws/addr/	W3C Last Call (LC) version of the WS-Addressing WSDL specification. Support for this namespace is deprecated.
http://schemas.xmlsoap.org/ws/2004/08/addressing	http://www.w3.org/Submission/ws-addressing/	W3C WS-Addressing Submission specification This specification is sometimes referred to as the 2004/08 specification. It combines the core, SOAP and WSDL aspects of WS-Addressing in a single specification.

This version of the product interoperates with each of the WS-Addressing specifications that are defined in the previous table. This interoperability results in the following behavior:

- Incoming web service messages that contain WS-Addressing message addressing properties are appropriately bound to SOAP, and WS-Addressing SOAP elements are appropriately deserialized to their WS-Addressing programming model representations according to the namespace in use.
- WS-Addressing programming model artifacts are appropriately serialized into SOAP elements, and the message addressing properties are bound to SOAP according to the namespace in use.
- Differing WS-Addressing semantics are adhered to, according to the WS-Addressing version currently in use.

Determining the WS-Addressing namespace of inbound messages

The WS-Addressing namespace of incoming web service messages is the namespace of the first WS-Addressing action message addressing property that is found. The runtime checks for an action message addressing property of the default namespace. If it does not find an action with the default namespace, it will then search for action message addressing properties for other addressing namespaces in an undefined order. The namespace of the WS-Addressing core specification in use is available to the target endpoint through the message context.

Determining the WS-Addressing namespace of outbound messages

WS-Addressing messages that are issued from this version of the product adopt the namespace that is associated with the destination endpoint reference. If this namespace is unknown, the message adopts the default WS-Addressing namespace.

This product provides a proprietary system programming interface (SPI) to change the namespace that is associated with an endpoint reference to any namespace in the supported set.

The WS-Addressing specification to use

best-practices: In most cases, use the default WS-Addressing specification that is supported by the product. You do not have to undertake any additional actions to use this specification. The following list gives examples of occasions where you must override the default namespace:

- When interoperating with an endpoint that does not support the default namespace, for example, an earlier version of the product.
- When a namespace other than the default is required. For example, when implementing a specification that uses a level of WS-Addressing other than the default.

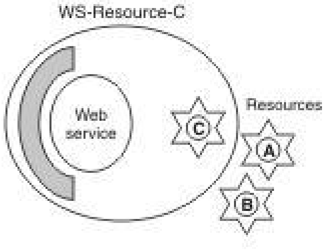
The W3C Last Call (LC) version of the WS-Addressing WSDL specification is deprecated. Use this specification only when you are interoperating with WebSphere Application Server 6.1 nodes that do not have fix pack V6.1.0.2 or later. Otherwise, use the W3C Candidate Recommendation version of the specification, or for JAX-WS applications, the WS-Addressing metadata specification.

Web Services Addressing application programming model

The Web Services Addressing (WS-Addressing) specification defines an endpoint reference that is represented in Extensible Markup Language (XML) by an EndpointReferenceType object that encapsulates information about the endpoint address as well as additional contextual information associated with the endpoint. Some services might be addressable by using a simple URI address, as is most typical in web services. Other services might require the use of an endpoint reference to address them, so that the additional contextual information associated with the endpoint is present in messages sent to the endpoint.

Examples of services that use WS-Addressing endpoint references include Web Services Resources and Web Services Notification message producers and message consumers, all of which have the notion of stateful resources associated with their endpoints. In these cases the endpoint reference not only contains the service address but also some data that is used to select the specific stateful resource instance for use in the processing of a web services message.

A WS-Resource is defined as the combination of a resource and a web service through which the resource is accessed. The following figure illustrates a web service, at http://www.example.com/service, and three resources, A, B, and C, which are accessed through the Web service. Three WS-Resources are therefore illustrated in the figure:



A WS-Resource is referenced by a WS-Addressing endpoint reference that uniquely identifies the WS-Resource, typically by containing an identifier of the resource component of the WS-Resource inside the EndpointReference ReferenceParameter element. In the previous example, WS-Resource-C is the combination of the web service and the resource that is identified by C, and a reference to WS-Resource-C might be as follows:

```
<wsa:EndpointReference>
    <wsa:Address>
        http://www.example.com/service
    </wsa:Address>
    <wsa:ReferenceParameters>
        <tns:SomeDisambiguatorElement>C</tns:SomeDisambiguatorElement>
    </wsa:ReferenceParameters>
    ...
</wsa:EndpointReference>
```

The WS-Addressing APIs provide the appropriate interfaces for implementing the previous pattern.

Web Services Addressing annotations

The WS-Addressing specification provides transport-neutral mechanisms to address web services and to facilitate end-to-end addressing. If you have a JAX-WS application you can use Java annotations in your code to specify WS-Addressing behavior at run time.

You can use WS-Addressing annotations to enable WS-Addressing support, to specify whether WS-Addressing information is required in incoming messages, to control the message exchange pattern the service supports, and to specify actions to be associated with a web service operation or fault response.

The following WS-Addressing annotations are supported in WebSphere Application Server. These annotations are defined in the JAX-WS 2.2 specification unless otherwise stated. The JAX-WS 2.2 specification supersedes and includes functions within the JAX-WS 2.1 specification. See the Java API for XML-Based Web Services 2.2 specification for full details.

javax.xml.ws.Action

Specifies the action that is associated with a web service operation.

- When following a bottom-up approach to developing JAX-WS web services, you can generate a WSDL document from Java application code using the **wsgen** command-line tool. However, for this attribute to be added to the WSDL operation, you must also specify the @Addressing annotation on the implementation class. The result in the generated WSDL document is that the Action annotations will have the `wsam:Action` attribute on the `input message` and `output message` elements of the `wsdl:operation`. Alternatively, if you do not want to use the @Addressing annotation you can supply your own WSDL document with the Action attribute already defined.
- When following a top-down approach to developing JAX-WS web services, you can generate Java application code from an existing WSDL document using the **wsimport** command-line tool. In such cases, the resulting Java code will contain the correct Action and FaultAction annotations.

If this action is not specified in either code annotations or in the WSDL document, the default action pattern as defined in the Web Services Addressing 1.0 Metadata specification is used. Refer to this specification for full details.

Note: Whilst the WebSphere Application Server runtime environment supports the deprecated `wsaw:Action` attribute, if you try to generate Java code from an old WSDL document containing the deprecated `wsaw:Action` attribute, this attribute will be ignored.

javax.xml.ws.FaultAction

Specifies the action that is added to a fault response. When you use this annotation with a particular method, the WS-Addressing FaultAction extension attribute is added to the fault element of the WSDL operation that corresponds to that method. For this attribute to be added to the

WSDL operation, you must also specify the Addressing annotation on the implementation class. If you do not want to use the Addressing annotation you can supply your own WSDL document with the Action attribute already defined. This annotation must be contained within an Action annotation.

WSDL documents generated from Java application code containing the WS-Addressing FaultAction annotation will have the `wsam:Action` attribute on the `fault message` element of the `wsdl:operation`.

Note: To ensure that any custom Exception classes you write are successfully mapped to the generated WSDL document, extend the java.lang.Exception class instead of the java.lang.RuntimeException class.

javax.xml.ws.soap.Addressing
Specifies that this service is to enable WS-Addressing support. You can use this annotation only on the service implementation bean; you cannot use it on the service endpoint interface.

com.ibm.websphere.wsaddressing.jaxws21.SubmissionAddressing
This annotation is part of the IBM implementation of the JAX-WS specification. This annotation specifies that this service is to enable WS-Addressing support for the 2004/08 WS-Addressing specification. You can use this annotation only on the service implementation bean; you cannot use it on the service endpoint interface.

For more information about the Addressing and SubmissionAddressing annotations, including code examples, see Enabling Web Services Addressing support for JAX-WS applications using addressing annotations.

The following example code uses the Action annotation to define the invoke operation to be invoked (input), and the action that is added to the response message (output). The example also uses the FaultAction annotation to specify the action that is added to a response message if a fault occurs:

```
@WebService(name = "Calculator")
public interface Calculator {
   ...
    @Action(
      input="http://calculator.com/inputAction",
      output="http://calculator.com/outputAction",
      fault = { @FaultAction(className=AddNumbersException.class,
                value="http://calculator.com/faultAction")
      }
   )
   public int add(int value1, int value2) throws AddNumbersException {
      return value1 + value2;
   }
}
```

If you use a tool to generate service artifacts from code, the WSDL tags that are generated from the preceding example are as follows:

```
<definitions targetNamespace="http://example.com/numbers" ...>
   ...
   <portType name="AddPortType">
      <operation name="Add">
         <input message="tns:AddInput" name="Parameters"
                        wsam:Action="http://calculator.com/inputAction"/>
         <output message="tns:AddOutput" name="Result"
                          wsam:Action="http://calculator.com/outputAction"/>
         <fault message="tns:AddNumbersException" name="AddNumbersException"
                         wsam:Action="http://calculator.com/faultAction"/>
      </operation>
    <portType>
    ...
<definitions>
```

Web Services Addressing security

It is essential that communications that use Web Services Addressing (WS-Addressing) are adequately secured and that a sufficient level of trust is established between the communicating parties. You can achieve secure communications through the signing of WS-Addressing message-addressing properties and the encryption of endpoint references.

Undertake these actions for both the supported addressing namespaces, http://www.w3.org/2005/08/addressing and http://schemas.xmlsoap.org/ws/2004/08/addressing, even if you intend to use only one of those namespaces.

Signing of WS-Addressing message-addressing properties

You can use an assembly tool to specify the message-addressing properties, and therefore the WS-Addressing message elements, that require signing, or that require signature verification on inbound requests. The receiver of the message might rely on the presence of this verifiable signature to determine that the outbound message originated from a trusted source. Similarly, the lack of a verifiable signature that is associated with the specified inbound message addressing properties causes the rejection of the message with a SOAP fault.

Encryption of endpoint references

You can encrypt endpoint references as part of the SOAP header or SOAP body. Alternatively, you can remove the need for encryption by not including sensitive information in the address or reference parameters properties of the endpoint reference.

Use of the synchronous message exchange pattern

This method applies to JAX-WS applications only.

If you do not secure the WS-Addressing information in the SOAP message by using one or more of the previous methods, and you do not have WS-Security enabled, the ReplyTo and FaultTo elements of the SOAP message could be used to send messages to a third party, potentially taking part in a Denial of Service attack. To prevent such attacks, apply a WS-Addressing policy type and configure it to specify synchronous messaging only. You should also enable WS-Policy so that this requirement is communicated to clients.

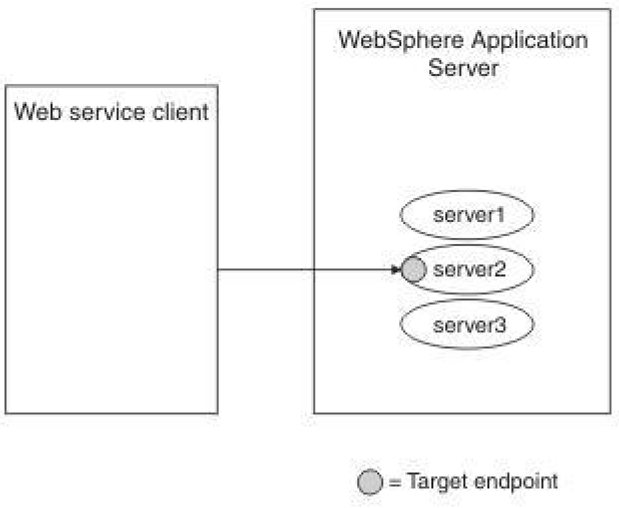
Web Services Addressing: firewalls and intermediary nodes

Using the Web Services Addressing (WS-Addressing) support in this product, you can create endpoint references that can be distributed across firewalls and intermediary nodes.

Using the WS-Addressing support, you can automatically generate endpoint references that represent endpoints on the node on which the references are generated. These endpoint references contain appropriate address information, based on the URL configured for the endpoint and any valid proxy configuration for the server that hosts the endpoint. Messages targeted at the endpoint reference from the client are routed to the endpoint through the appropriate intermediary node or nodes, as described in the following topology scenarios.

Direct connection

In this topology, there is no intermediary node. The client communicates directly with the server that hosts the target endpoint. In this topology, the WS-Addressing APIs automatically generate the appropriate endpoint reference address, based on the URL configured for the web service module. This scenario is illustrated in the following diagram.



Endpoint references created by using the standard JAX-WS API are not workload managed.

HTTP server, such as IBM HTTP Server

In this topology, the IBM WS-Addressing API automatically generates the appropriate endpoint reference address based on the URL prefix of the HTTP server that is configured for the target web service module. The client communicates with the HTTP server, which then routes the client requests to a specific server based on the HTTP server configuration.

Web Services Addressing and the service integration bus

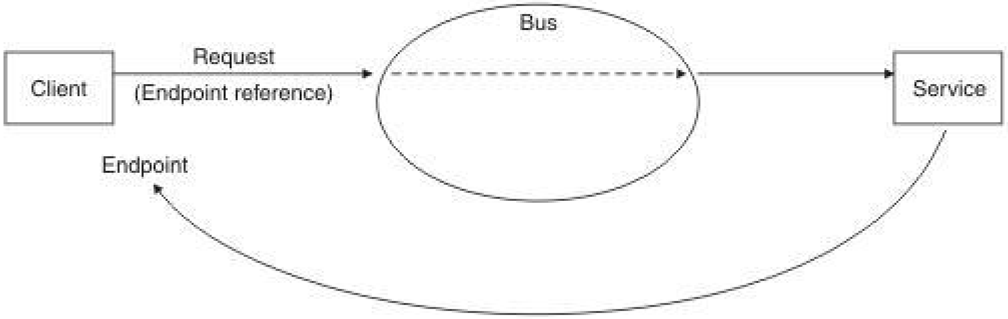
If you are using the Web Services Addressing (WS-Addressing) support, the presence of a service integration bus can affect the routing of messages. If you are also using a firewall, you might have to complete some additional configuration.

In the following scenarios, the client must conform to the WS-Addressing specification.

One-way messaging scenario

The path taken by one-way messages is as follows:

1. The client sends a request, containing an endpoint reference specifying the endpoint to which replies are sent, to the service integration bus. This request is a one way request, so the client does not wait for a response.
2. The bus passes the message intact to the web service.
3. The web service sends a response directly to the endpoint that is specified in the request.

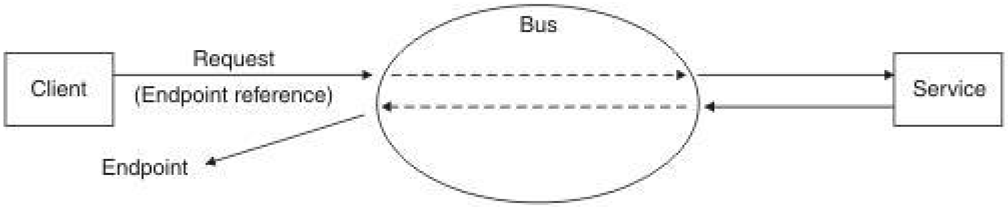


This scenario works if messages can flow directly from the Web service to the endpoint. If you have a configuration that does not support direct message flow, for example if you have a firewall, you must create handlers that can redirect the message as required.

Request-response messaging scenario

For request-response scenarios, the messages take the following path:

1. The client sends a request, containing an endpoint reference specifying the endpoint to which replies are sent, to the service integration bus.
2. The bus passes the message intact to the web service, as a synchronous request. As the message leaves the bus, the endpoint reference is replaced with the anonymous URI listed in the WS-Addressing specification. This step ensures that the web service does not send a response directly to the endpoint.
3. The web service sends a response back to the bus, as part of the synchronous interaction.
4. As the message leaves the bus, the anonymous URI is replaced with the original endpoint reference, enabling the bus to pass the message to the endpoint.



Web Services Addressing APIs

This product provides interfaces at the application programming level to enable application developers, including developers of Web Services Resource Framework applications, to create references to, and to target, web service resource instances. If you are a system programmer, you can use some these interfaces with the Web Services Addressing (WS-Addressing) system programming interfaces.

JAX-WS 2.1 APIs

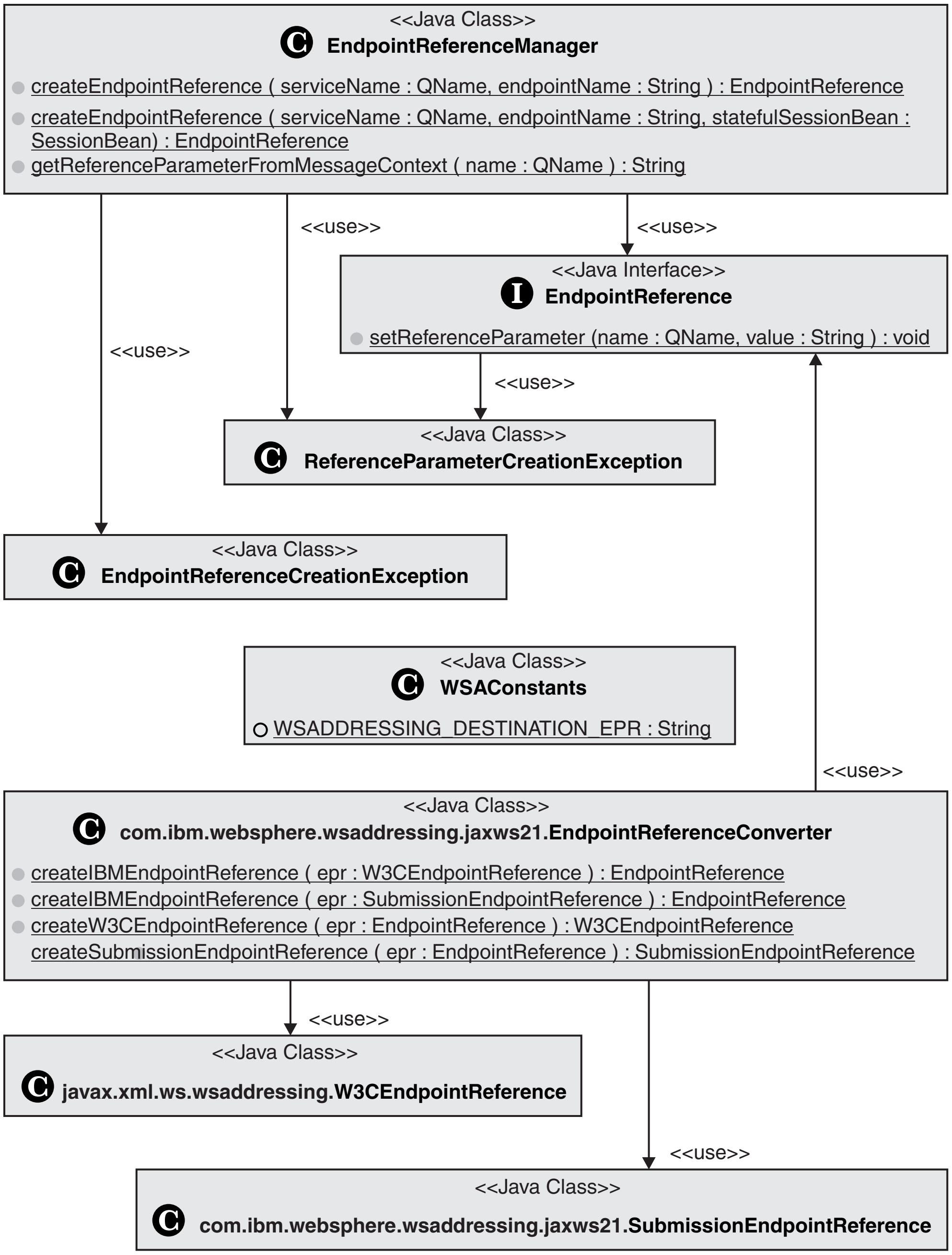
The standard JAX-WS 2.1 APIs in this product are contained in the javax.xml.ws.wsaddressing package. Refer to the JAX-WS 2.1 API documentation for more information about these APIs.

The implementation of the standard JAX-WS 2.1 APIs in this product also contains application programming interfaces, in the com.ibm.websphere.wsaddressing.jaxws21 package. These APIs are described in more detail in the generated API documentation in this information center. These APIs allow you to achieve the following objectives by using specific classes:

- To represent endpoints that conform to the 2004/08 WS-Addressing specification, use the com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReference class.
- To create a SubmissionEndpointReference instance to represent 2004/08 endpoints in web services other than the one generating the endpoint reference, use the com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReferenceBuilder class.
- To convert EndpointReference instances created by using the IBM proprietary WS-Addressing API into either W3CEndpointReference or SubmissionEndpointReference instances, use the com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceCoverter class. This class can also be used to reverse the conversion of EndpointReference instances.
- To enable WS-Addressing on clients, use the com.ibm.websphere.wsaddressing.jaxws21.SubmissionAddressingFeature class, and an annotation, @SubmissionAddressing, for enabling WS-Addressing on servers.

IBM proprietary WS-Addressing APIs

These application programming interfaces are contained in the com.ibm.websphere.wsaddressing package and are summarized in the following diagram. The diagram also shows the following classes from the JAX-WS 2.1 API: com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceConverter, javax.xml.ws.wsaddressing.W3CEndpointReference and com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReference.



<<Java Class>>
EndpointReferenceManager
createEndpointReference (serviceName : QName, endpointName : String) : EndpointReference
createEndpointReference (serviceName : QName, endpointName : String, statefulSessionBean : SessionBean) : EndpointReference
getReferenceParameterFromMessageContext (name : QName) : String
<<use>>
<<use>>
<<use>>
<<Java Interface>>
EndpointReference
setReferenceParameter (name : QName, value : String) : void
<<use>>
<<Java Class>>
ReferenceParameterCreationException
<<Java Class>>
EndpointReferenceCreationException
<<Java Class>>
WSAConstants
WSADDRESSING_DESTINATION_EPR : String
<<use>>
<<Java Class>>
com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceConverter
createIBMEndpointReference (epr : W3CEndpointReference) : EndpointReference
createIBMEndpointReference (epr : SubmissionEndpointReference) : EndpointReference
createW3CEndpointReference (epr : EndpointReference) : W3CEndpointReference
createSubmissionEndpointReference (epr : EndpointReference) : SubmissionEndpointReference
<<use>>
<<Java Class>>
javax.xml.ws.wsaddressing.W3CEndpointReference
<<use>>
<<Java Class>>
com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReference

These interfaces provide the following features:

- A mechanism for creating a com.ibm.websphere.wsaddressing.EndpointReference instance to represent a WS-Addressing endpoint reference by using the com.ibm.websphere.wsaddressing.EndpointReferenceManager.createEndpointReference interface.
-

Note: A deprecated class, com.ibm.websphere.wsaddressing.EndpointReferenceCoverter, for converting EndpointReference instances into deprecated classes com.ibm.websphere.wsaddressing.W3CEndpointReference or com.ibm.websphere.wsaddressing.SubmissionEndpointReferences, for use in JAX-WS applications.

These classes are deprecated in favour of the JAX-WS 2.1 classes of the same name (EndpointReferenceConverter, SubmissionEndpointReference, and W3CEndpointReference) contained in the com.ibm.websphere.wsaddressing.jaxws21 and javax.xml.ws.wsaddressing.jaxws21 packages, as shown on the diagram.

- A method, com.ibm.websphere.wsaddressing.EndpointReference.setReferenceParameter, to enable you to associate reference parameters with an EndpointReference instance.
- An interface to enable a client to configure its BindingProvider request context, or Stub or Call object, based on an EndpointReference instance. All invocations on the BindingProvider, Stub or Call object are subsequently targeted at the endpoint that is represented by the EndpointReference instance. To achieve this behavior, set the com.ibm.websphere.wsaddressing.WSAConstants.WSADDRESSING_DESTINATION_EPR property on the BindingProvider request context, or Stub or Call object, to the appropriate EndpointReference instance.
- A mechanism for acquiring individual reference parameters that are associated with the incoming message context, to correlate the message to a specific resource instance through the com.ibm.websphere.EndpointReferenceManager.getReferenceParameterFromMessageContext interface.

IBM proprietary Web Services Addressing SPIs

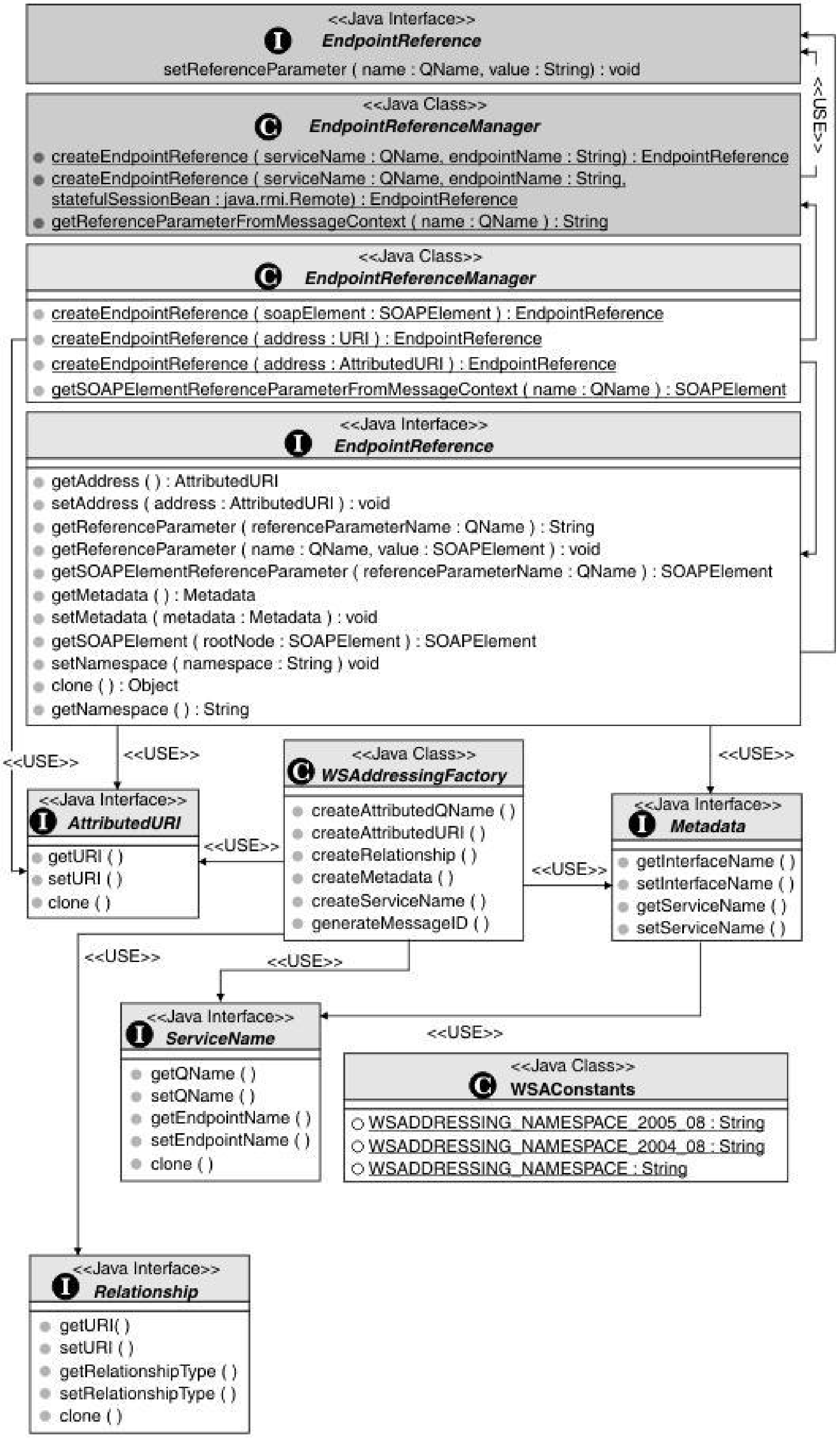
The IBM proprietary Web Services Addressing (WS-Addressing) system programming interfaces (SPIs) extend the IBM proprietary WS-Addressing application programming interfaces (APIs) to enable you to create and reason about the contents of endpoint references and other WS-Addressing artifacts, and to set or retrieve WS-Addressing message-addressing properties (MAPs) on or from web service messages.

You cannot use the standard JAX-WS API classes with these proprietary SPIs. However, you can convert endpoint references created by using the standard JAX-WS API classes to instances of the proprietary com.ibm.websphere.wsaddressing.EndpointReference class, using the com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceConverter class. You can then use these converted endpoint references with the proprietary SPIs.

The programming interfaces in this topic are described in more detail in the IBM WS-Addressing SPI documentation.

Creating, refining, and reasoning about the contents of endpoint references

The proprietary SPIs for creating, refining, and reasoning about the contents of endpoint references are contained in the com.ibm.wsspi.wsaddressing package and are summarized in the following illustration (the first two interfaces are proprietary API interfaces that are extended by the SPIs):


<<Java Interface>>
I
EndpointReference
setReferenceParameter (name : QName, value : String) : void
<<Java Class>>
C
EndpointReferenceManager
createEndpointReference (serviceName : QName, endpointName : String) : EndpointReference
createEndpointReference (serviceName : QName, endpointName : String, statefulSessionBean : java.rmi.Remote) : EndpointReference
getReferenceParameterFromMessageContext (name : QName) : String
<<USE>>
<<Java Class>>
C
EndpointReferenceManager
createEndpointReference (soapElement : SOAPElement) : EndpointReference
createEndpointReference (address : URI) : EndpointReference
createEndpointReference (address : AttributedURI) : EndpointReference
getSOAPElementReferenceParameterFromMessageContext (name : QName) : SOAPElement
<<Java Interface>>
I
EndpointReference
getAddress ( ) : AttributedURI
setAddress (address : AttributedURI) : void
getReferenceParameter (referenceParameterName : QName) : String
getReferenceParameter (name : QName, value : SOAPElement) : void
getSOAPElementReferenceParameter (referenceParameterName : QName) : SOAPElement
getMetadata ( ) : Metadata
setMetadata (metadata : Metadata) : void
getSOAPElement (rootNode : SOAPElement) : SOAPElement
setNamespace (namespace : String) void
clone ( ) : Object
getNamespace ( ) : String
<<USE>>
<<USE>>
<<USE>>
<<Java Class>>
C
WSAddressingFactory
createAttributedQName ( )
createAttributedURI ( )
createRelationship ( )
createMetadata ( )
createServiceName ( )
generateMessageID ( )
<<Java Interface>>
I
AttributedURI
getURI ( )
setURI ( )
clone ( )
<<USE>>
<<USE>>
<<Java Interface>>
I
Metadata
getInterfaceName ( )
setInterfaceName ( )
getServiceName ( )
setServiceName ( )
<<USE>>
<<USE>>
<<Java Interface>>
I
ServiceName
getQName ( )
setQName ( )
getEndpointName ( )
setEndpointName ( )
clone ( )
<<USE>>
<<Java Class>>
C
WSAConstants
○ WSADDRESSING_NAMESPACE_2005_08 : String
○ WSADDRESSING_NAMESPACE_2004_08 : String
○ WSADDRESSING_NAMESPACE : String
<<Java Interface>>
I
Relationship
getURI( )
setURI ( )
getRelationshipType ( )
setRelationshipType ( )
clone ( )

The SPI extends the proprietary WS-Addressing com.ibm.websphere.wsaddressing.EndpointReference API to provide a number of additional methods through the com.ibm.wsspi.wsaddressing.EndpointReference interface. You can cast instances of com.ibm.websphere.wsaddressing.EndpointReference to com.ibm.wsspi.wsaddressing.EndpointReference to access these additional functions.

Similarly, the SPI com.ibm.wsspi.wsaddressing.EndpointReferenceManager extends the set of functions that are provided in the com.ibm.websphere.wsaddressing.EndpointReferenceManager API.

You can complete the following actions by using the additional methods that are provided by the EndpointReference and EndpointReferenceManager SPIs:

Create endpoint references

Create EndpointReference objects by specifying the URI of the endpoint that the EndpointReference object is to represent, by using the createEndpointReference(URI) operation, or the EndpointReferenceManager.createEndpointReference(AttributedURI) operation. These methods differ from the createEndpointReference method that is provided at the API level, in that they do not automatically generate the URI for the EndpointReference instance. You might use these methods when you know that the URI of the endpoint is stable, for example in a test environment with no deployment restrictions.

Map between XML and Java representations of an endpoint reference

You can serialize instances of the EndpointReference interface to their corresponding SOAP element instances by using the EndpointReference.getSOAPElement operation. Conversely, you can deserialize SOAP elements of type EndpointReferenceType into their corresponding EndpointReference Java representation, by using the EndpointReference.createEndpointReference(SOAPElement) operation. You might find these serialization and deserialization interfaces useful if you are creating custom binders for types that contain EndpointReference instances.

Use more complex reference parameter types

The proprietary interfaces that are provided at the API level are restricted to reference parameters of type xsd:string to allow for a simpler programming model. The SPIs extend this support to allow reference parameters of type <xsd:any>. The EndpointReference interface provides mechanisms for getting and setting reference parameters as SOAP elements. Additionally, the EndpointReferenceManager class provides the getSOAPElementReferenceParameterFromMessageContext operation, which enables receiving endpoints to acquire reference parameters that are not of type String from the incoming message.

Note: When invoking a service with an EndpointReference object that contains a reference parameter, you must create the reference parameter by using a complete QName object, with all parts present: namespace, localpart, and prefix. If the QName object is not complete, service invocations fail.

Set and reason about endpoint reference contents

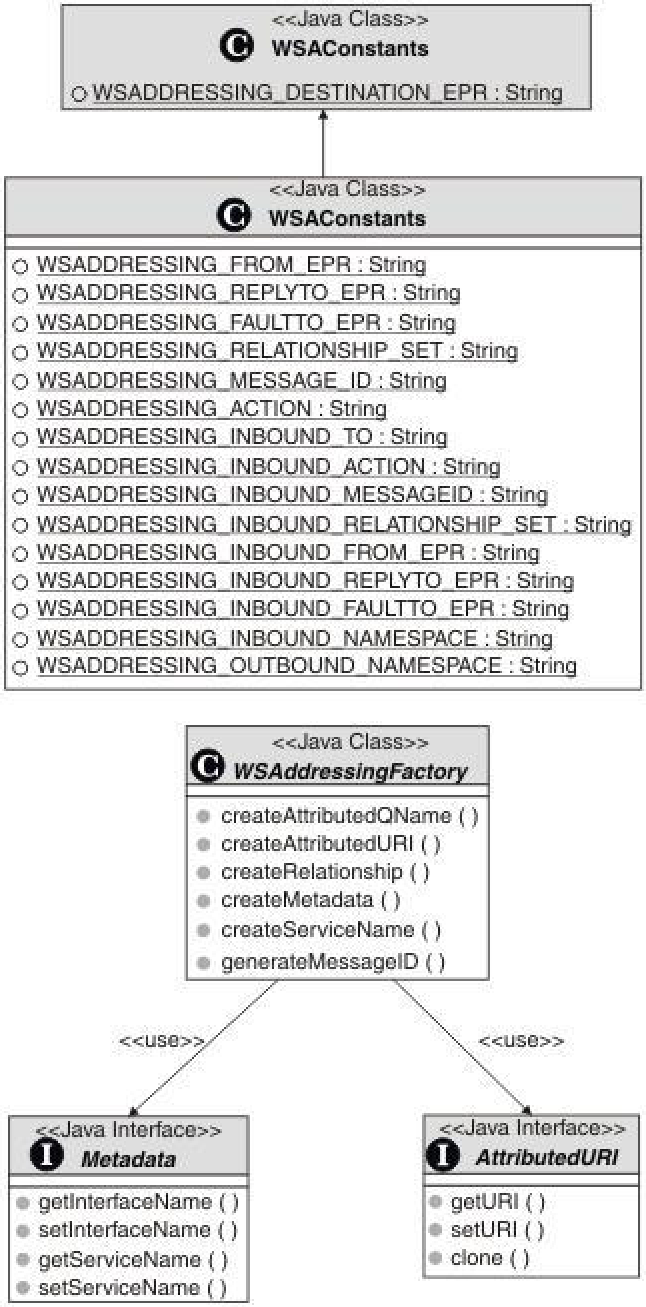
The EndpointReference interface provides operations for you to set and reason about the contents of an EndpointReference instance, such as its WS-Addressing address and metadata properties. Additional interfaces are provided to represent the artifacts making up an endpoint reference: Metadata, AttributedURI, and ServiceName. You create instances of these interfaces by using operations that are provided by the proprietary WSAddressingFactory class.

Acquire and change the supported namespace

The WS-Addressing support in this product supports multiple namespaces. The setNamespace and getNamespace operations that are provided on the proprietary EndpointReference interface enable you to change and acquire the namespace that is associated with a particular EndpointReference object. Serialization to SOAP elements is in accordance with the namespace of the EndpointReference object. By default, the namespace of the destination endpoint reference (the endpoint reference set as the com.ibm.websphere.wsaddressing.WSAConstants.WSADDRESSING_DESTINATION_EPR property on the JAX-WS BindingProvider object request context, or the JAX-RPC Stub or Call object), defines the namespace of the message-addressing properties of the message.

Setting and Retrieving WS-Addressing message-addressing properties

The IBM proprietary WS-Addressing SPI provides a number of constants that identify JAX-WS or JAX-RPC properties that you can use to set WS-Addressing MAPs on outbound messages, and message context properties that you can use to retrieve MAPs on inbound messages. These constants are shown in the following diagram in the com.wsspi.wsaddresssing.WSAConstants class. The diagram also shows the interfaces that are required for generating instances of the appropriate property value types AttributedURI and Relationship. The first WSAConstants interface is a proprietary API interface.



<<Java Class>>
WSAConstants
WSADDRESSING_DESTINATION_EPR : String
<<Java Class>>
WSAConstants
WSADDRESSING_FROM_EPR : String
WSADDRESSING_REPLYTO_EPR : String
WSADDRESSING_FAULTTO_EPR : String
WSADDRESSING_RELATIONSHIP_SET : String
WSADDRESSING_MESSAGE_ID : String
WSADDRESSING_ACTION : String
WSADDRESSING_INBOUND_TO : String
WSADDRESSING_INBOUND_ACTION : String
WSADDRESSING_INBOUND_MESSAGEID : String
WSADDRESSING_INBOUND_RELATIONSHIP_SET : String
WSADDRESSING_INBOUND_FROM_EPR : String
WSADDRESSING_INBOUND_REPLYTO_EPR : String
WSADDRESSING_INBOUND_FAULTTO_EPR : String
WSADDRESSING_INBOUND_NAMESPACE : String
WSADDRESSING_OUTBOUND_NAMESPACE : String
<<Java Class>>
WSAddressingFactory
createAttributedQName ( )
createAttributedURI ( )
createRelationship ( )
createMetadata ( )
createServiceName ( )
generateMessageID ( )
<<use>>
<<use>>
<<Java Interface>>
Metadata
getInterfaceName ( )
setInterfaceName ( )
getServiceName ( )
setServiceName ( )
<<Java Interface>>
AttributedURI
getURI ( )
setURI ( )
clone ( )

Setting WS-Addressing message-addressing properties on outbound messages:

You can add WS-Addressing message information headers to outgoing messages by setting the appropriate properties on the JAX-WS BindingProvider object request context, or the JAX-RPC Stub or Call object, prior to invoking a message with the BindingProvider, Stub, or Call object. The following table summarizes the relevant properties and their types.

Table 65. Outbound properties that you can set on the BindingProvider object request context (or the Stub or Call object). The table lists the different property names, their Java types, their abstract WS-Addressing MAP names and their default values.

Property name (of type String)	**Java type of property value**	**Abstract WS-Addressing MAP name or names, using the notational convention of the W3C XML Information Set**	**Default value**
WSADDRESSING_ DESTINATION_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[destination] URI [reference parameters]* (any)	Not set Note that this property comes from the API.
WSADDRESSING_ FROM_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[source endpoint]	Not set
WSADDRESSING_ REPLYTO_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[reply endpoint]	Either 'none', if the message is a one-way message with no reply, or not set. For two-way asynchronous messages in JAX-WS applications, this property is generated automatically. If, in this situation, you attempt to set this property, a javax.xml.ws.WebServiceException exception is thrown. This exception is also thrown for two-way synchronous messages that do not use the anonymous URI.
WSADDRESSING_ FAULTTO_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[fault endpoint]	Not set If you attempt to set this property for two-way asynchronous messages in JAX-WS applications, a javax.xml.ws.WebServiceException exception is thrown. This exception is also thrown for two-way synchronous messages that do not use the anonymous URI.
WSADDRESSING_ RELATIONSHIP_SET	java.util.Set containing instances of com.ibm.wsspi.wsaddressing. Relationship	[relationship]	Not set
WSADDRESSING_ MESSAGE_ID	com.ibm.wsspi.wsaddressing. AttributedURI	[message id]	Generated and set to a unique value
WSADDRESSING_ ACTION	com.ibm.wsspi.wsaddressing. AttributedURI	[action]	Generated and set, according to the WS-Addressing specification
WSADDRESSING_ OUTBOUND_ NAMESPACE	String	none	The WS-Addressing namespace of the WSADDRESSING_ DESTINATION_EPR property, if specified, otherwise the default namespace

Retrieving WS-Addressing message-addressing properties from inbound messages:

WS-Addressing message information headers that correspond to the last inbound message are available from the inbound properties that are defined in the WSAConstants class. The following table summarizes the available inbound properties. You acquire reference parameters from the message context by using the proprietary EndpointReferenceManager.getReferenceParameter interface.

Table 66. Inbound properties that you can acquire from the message context. The table lists the different property names, their Java types and equivalent abstract WS-Addressing MAP names.

Message context property name (of type String)	Java type of property value	Abstract WS-Addressing MAP name, using the notational convention of the W3C XML Information Set
WSADDRESSING_INBOUND_TO	com.ibm.wsspi.wsaddressing. AttributedURI	[destination]
No specific property. Use the EndpointReferenceManager. getReferenceParameter(QName name) method to obtain the associated MAP.	Any	[reference parameters]*
WSADDRESSING_INBOUND_FROM_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[source endpoint]
WSADDRESSING_INBOUND_REPLYTO_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[reply endpoint]
WSADDRESSING_INBOUND_FAULTTO_EPR	com.ibm.websphere.wsaddressing. EndpointReference	[fault endpoint]
WSADDRESSING_INBOUND_RELATIONSHIP	java.util.Set containing instances of com.ibm.wsspi.wsaddressing. Relationship	[relationship]
WSADDRESSING_INBOUND_MESSAGE_ID	com.ibm.wsspi.wsaddressing. AttributedURI	[message id]
WSADDRESSING_INBOUND_ACTION	com.ibm.wsspi.wsaddressing. AttributedURI	[action]
WSADDRESSING_INBOUND_NAMESPACE	String	The WS-Addressing namespace of the incoming message

Web Services Resource Framework support

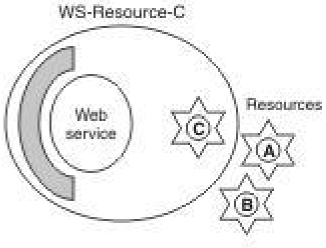
The Web Services Resource Framework (WSRF) support in WebSphere Application Server provides the environment for web service applications that follow the OASIS WSRF specifications.

WSRF overview

Web service interfaces often need to provide stateful interactions with the clients of the service. For example, a web service interface such as a shopping cart, where the result of one operation influences the carrying out of the succeeding operations. The OASIS Web Services Resource Framework (WSRF) defines a generic framework for modelling and accessing stateful resources using web services, so that the definition and implementation of a service and the integration and management of multiple services is easier.

WSRF introduces the concept of an XML document description, called the *resource properties document schema*, which is referenced by the WSDL description of a web service and which explicitly describes a view of the state of the resource with which the client interacts. A service described in this way is called a *WS-Resource*.

A WS-Resource is defined as the combination of a resource and a web service through which the resource is accessed. The following figure illustrates a web service, at http://www.example.com/service, and three resources, A, B, and C, which are accessed through the Web service. Three WS-Resources are therefore illustrated in the figure:



A WS-Resource is referenced by a WS-Addressing endpoint reference that uniquely identifies the WS-Resource, typically by containing an identifier of the resource component of the WS-Resource inside the EndpointReference ReferenceParameter element. In the previous example, WS-Resource-C is the combination of the web service and the resource that is identified by C, and a reference to WS-Resource-C might be as follows:

```
<wsa:EndpointReference>
    <wsa:Address>
        http://www.example.com/service
    </wsa:Address>
    <wsa:ReferenceParameters>
        <tns:SomeDisambiguatorElement>C</tns:SomeDisambiguatorElement>
    </wsa:ReferenceParameters>
    ...
</wsa:EndpointReference>
```

Each such WS-Resource has a resource property document (an XML instance document) that describes a view of the state of the resource. The WSDL for a WS-Resource identifies the XML schema that describes the type of the resource property document through a ResourceProperties attribute of the wsdl:PortType element. By specifying this standard WSDL extension for the resource properties document schema, WSRF enables the definition of simple, generic messages that interact with the WS-Resource.

For example, consider a Printer WS-Resource that has the following resource properties document schema:

```
<?xml version="1.0"?>
<xsd:schema ...
   xmlns:pr="http://example.org/printer.xsd"
   targetNamespace="http://example.org/printer.xsd" >
 <xsd:element name="printer_properties">
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element ref="pr:printer_name" />
        <xsd:element ref="pr:queued_job_count" />
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
 ...
</schema>
```

The WSDL PortType element for such a WS-Resource declares the Resource Properties Document type as follows:

```
<wsdl:portType xmlns:pr="http://example.org/printer.xsd"
               xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
               name="Printer" wsrf-rp:ResourceProperties="pr:printer_properties">
```

Each WS-Resource has a unique, logical resource properties document instance that is a view of the state of the resource. The WS-ResourceProperties specification describes the interoperable protocol messages that a WS-Resource can implement to get, set, or query the state of the resource by operating on the resource properties document. Some of these operations affect the resource properties document as a whole, and some of them operate on one or more elements within the document (the individual resource properties, for example pr:printer_name). Each WS-Resource can have a finite lifecycle and can be created and destroyed; the WS-ResourceLifetime specification describes the interoperable protocol messages that a WS-Resource can implement to destroy itself or to alter its termination time.

For more information about WSRF, refer to the WSRF Primer document published by the OASIS Technical Committee.

WSRF Programming Model

The WSRF specifications define only the protocol messages and the semantic behavior that is expected of a WS-Resource when it processes these messages; the specifications do not prescribe the means to implement WS-Resource objects. WSRF is primarily an application-level protocol and the tools for implementing WS-Resources are the same tools that are used for implementing any other type of web service. WSRF uses WS-Addressing endpoint references and the application programming model for WS-Resources is similar to the model for any web service that uses WS-Addressing.

WSRF extends the WebSphere Application Server WS-Addressing programming model in two ways, which differentiate a WS-Resource from a generic resource that is accessed through a web service by using WS-Addressing:

- WSRF requires the ResourceProperties attribute on the wsdlPortType element. This attribute declares that the portType element is implemented by a WS-Resource rather than a generic web service. The WS-Resource must declare which WSRF operations it supports by copying those operations into the portType element of its WSDL definition. The WS-Resource is free to choose any implementation strategy to represent the stateful resource and to process the WSRF messages; you can implement a resource using a simple Java class, a stateless session enterprise bean, an entity bean backed by a relational database, a Service Data Object (SDO), and so on.
- WSRF defines a hierarchy of Java BaseFault types.

Web Services Resource Framework base faults

The Web Services Resource Framework (WSRF) provides a recommended basic fault message element type from which you can derive all service-specific faults. The advantage of a common basic type is that all faults can, by default, contain common information. This behavior is useful in complex systems where faults might be systematically logged, or forwarded through several layers of software before being analyzed.

The common information includes the following items:

- A mandatory timestamp.
- An element that can be used to indicate the originator of the fault.
- Other elements that can describe and classify the fault.

The following two standard faults are defined for use with every WSRF operation:

ResourceUnkownFault
This fault is used to indicate that the WS-Resource is not known by the service that receives the message.

ResourceUnavailableFault
This fault is used to indicate that the web service is active, but temporarily unable to provide access to the resource.

The following XML fragment shows an example of a base fault element:

```
<wsrf-bf:BaseFault>
   <wsrf-bf:Timestamp>2005-05-31T12:00:00.000Z</wsrf-bf:Timestamp>
   <wsrf-bf:Originator>
     <wsa:Address>
       http://www.example.org/Printer
     </wsa:Address>
     <wsa:ReferenceParameters>
       <pr:pr-id>P1</pr:pr-id>
     </wsa:ReferenceParameters>
   </wsrf-bf:Originator>
   <wsrf-bf:Description>Offline for service maintenance</wsrf-bf:Description>
   <wsrf-bf:FaultCause>OFFLINE</wsrf-bf:FaultCause>
</wsrf-bf:BaseFault>
```

Important: The elements and classes that are discussed in the rest of this topic apply to JAX-RPC applications only. If your application uses JAX-WS, use the artifacts that are generated, for example by the wsimport tool, from the application WSDL document and XML schema that define and use the specific BaseFault type.

The BaseFault class

For JAX-RPC applications, WebSphere Application Server provides Java code mappings for all the base fault element types that are defined by the WSRF specifications, forming an exception hierarchy where each Java exception extends the com.ibm.websphere.wsrf.BaseFault class. Each fault class follows a similar pattern.

For example, the BaseFault class defines the following two constructors:

```
package com.ibm.websphere.wsrf;
public class BaseFault extends Exception
{
    public BaseFault()
    {
       ...
    }
    public BaseFault(EndpointReference originator,
                     ErrorCode errorCode,
                     FaultDescription[] descriptions,
                     IOSerializableSOAPElement faultCause,
                     IOSerializableSOAPElement[] extensibilityElements,
                     Attribute[] attributes)
    {
       ...
    }
    ...
}
```

The IOSerializableSOAPElement class

Because the BaseFault class extends the java.lang.Exception class, the BaseFault class must implement the java.io.Serializable interface. To meet this requirement, all properties of a BaseFault instance must be serializable. Because the javax.xml.soap.SOAPElement class is not serializable, WebSphere Application Server provides an IOSerializableSOAPElement class, which you can use to wrap a javax.xml.soap.SOAPElement instance to provide a serializable form of that instance.

Create an IOSerializableSOAPElement instance by using the IOSerializableSOAPElementFactory class, as follows:

```
// Get an instance of the IOSerializableSOAPElementFactory class
IOSerializableSOAPElementFactory factory = IOSerializableSOAPElementFactory.newInstance();

// Create an IOSerializableSOAPElement from a javax.xml.soap.SOAPElement
IOSerializableSOAPElement serializableSOAPElement = factory.createElement(soapElement);

// You can retrieve the wrapped SOAPElement from the IOSerializableSOAPElement
SOAPElement soapElement = serializableSOAPElement.getSOAPElement();
```

Any application-specific BaseFault instances must also adhere to this serializable requirement.

Application-specific faults

Applications can define their own extensions to the BaseFault element. Use XML type extensions to define a new XML type for the application fault that extends the BaseFaultType element. For example, the following XML fragment creates a new PrinterFaultType element:

```
 <xsd:complexType name="PrinterFaultType">
   <xsd:complexContent>
     <xsd:extension base="wsrf-bf:BaseFaultType"/>
   </xsd:complexContent>
 </xsd:complexType>
```

The following example shows how a web service application, whose WSDL definition might define a print operation that declares two wsdl:fault messages, constructs a PrinterFault object:

```
import com.ibm.websphere.wsrf.BaseFault;
import com.ibm.websphere.wsrf.*;
import javax.xml.soap.SOAPFactory;
...
    public void print(PrintRequest req) throws PrinterFault, ResourceUnknownFault
    {
        // Determine the identity of the target printer instance.
        PrinterState state = PrinterState.getState ();
        if (state.OFFLINE)
        {
            try
            {
                // Get an instance of the SOAPFactory
                SOAPFactory soapFactory = SOAPFactory.newInstance();

                // Create the fault cause SOAPElement
                SOAPElement faultCause = soapFactory.createElement("FaultCause");
                faultCase.addTextNode("OFFLINE");

                // Get an instance of the IOSerializableSOAPElementFactory
                IOSerializableSOAPElementFactory factory = IOSerializableSOAPElementFactory.newInstance();

                // Create an IOSerializableSOAPElement from the faultCause SOAPElement
                IOSerializableSOAPElement serializableFaultCause = factory.createElement(faultCause);

                FaultDescription[] faultDescription = new FaultDescription[1];
                faultDescription[0] = new FaultDescription("Offline for service maintenance");
                throw new PrinterFault(
                         state.getPrinterEndpointReference(),
                         null,
                         faultDescription,
                         serializableFaultCause,
                         null,
                         null);
            }
            catch (SOAPException e)
            {
               ...
            }
        }
        ...
```

The following code shows how base fault hierarchies are handled as Java exception hierarchies:

```
import com.ibm.websphere.wsrf.BaseFault;
import com.ibm.websphere.wsrf.*;
...
try
{
   printer1.print(job1);
}
catch (ResourceUnknownFault exc)
{
   System.out.println("Operation threw the ResourceUnknownFault");
}
catch (PrinterFault exc)
{
   System.out.println("Operation threw PrinterFault");
}
catch (BaseFault exc)
{
   System.out.println("Exception is another BaseFault");
}
catch (Exception exc)
{
   System.out.println("Exception is not a BaseFault");
}
```

Custom binders

When you define a new application-level base fault, for example the PrinterFault fault with the PrinterFaultType type shown previously, you must provide a custom binder to define how the web services run time serializes the Java class into an appropriate XML message, and conversely how to deserialize an XML message into an instance of the Java class.

The custom binder must implement the com.ibm.wsspi.webservices.binding.CustomBinder interface. Package the binder in a Java archive (JAR) file along with a declarative metadata file, `CustomBindingProvider.xml`, in the `/META-INF/services` directory of the JAR file. This metadata file defines the relationship between the custom binder, the Java BaseFault implementation and the BaseFault type. For example, you might define a custom binder called PrinterFaultTypeBinder, to map between the XML PrinterFaultType element and its Java implementation, PrinterFault, as follows:

```
<customdatabinding:provider
   xmlns:customdatabinding="http://www.ibm.com/webservices/customdatabinding/2004/06"
   xmlns:pr="http://example.org/printer.xsd"
   xmlns="http://www.ibm.com/webservices/customdatabinding/2004/06">
 <mapping>
  <xmlQName>pr:PrinterFaultType</xmlQName>
  <javaName>PrinterFault</javaName>
  <qnameScope>complexType</qnameScope>
  <binder>PrinterFaultTypeBinder</binder>
 </mapping>
</customdatabinding:provider>
```

The BaseFaultBinderHelper class

WebSphere Application Server provides a BaseFaultBinderHelper class, which provides support for serializing and deserializing the data that is specific to a root BaseFault class, which all specialized BaseFault classes must extend. If a custom binder uses the BaseFaultBinderHelper class, the custom binder then needs to provide only the additional logic for serializing and deserializing the extended BaseFault data.

The following code shows how you can implement a custom binder for the PrinterFaultType element to take advantage of the BaseFaultBinderHelper class support:

```
import com.ibm.wsspi.wsrf.BaseFaultBinderHelper;
import com.ibm.wsspi.wsrf.BaseFaultBinderHelperFactory;
import com.ibm.wsspi.webservices.binding.CustomBinder;
import com.ibm.wsspi.webservices.binding.CustomBindingContext;
...

public PrinterFaultTypeBinder implements CustomBinder
{
   // Get an instance of the BaseFaultBinderHelper
   private BaseFaultBinderHelper baseFaultBinderHelper = BaseFaultBinderHelperFactory.getBaseFaultBinderHelper();

   public SOAPElement serialize(Object data, SOAPElement rootNode, CustomBindingContext context) throws SOAPException
   {
     // Serialize the BaseFault specific data
     baseFaultBinderHelper.serialize(rootNode, (BaseFault)data);

     // Serialize any PrinterFault specific data
     ...

     // Return the serialized PrinterFault
     return rootNode;
   }

   public Object deserialize(SOAPElement rootNode, CustomBindingContext context) throws SOAPException
   {
     // Create an instance of a PrinterFault
     PrinterFault printerFault = new PrinterFault();

     // Deserialize the BaseFault specific data - any additional data that
     // forms the PrinterFault extension will be returned as a SOAPElement[].
     SOAPElement[] printerFaultElements = baseFaultBinderHelper.deserialize(printerFault, rootNode);

     // Deserialize the PrinterFault specific data contained within the printerFaultElements SOAPElement[]
     ...

     // Return the deserialized PrinterFault
     return printerFault;
   }

  ...

}
```

Web Services Resource Framework resource property and lifecycle operations

The Web Services Resource Framework (WSRF) contains specifications that describe the operations that a Web Services Resource (WS-Resource) can implement to get, set, or query the state of the resource by operating on the resource properties document.

For a complete description of all the standard property and lifetime operations that are defined by the Web Services Resource Framework (WSRF), see the WS-ResourceProperties and WS-ResourceLifetime specifications. The principle WSRF operations that a Web Services Resource (WS-Resource) can support are described in the following table.

Table 67. Principle WSRF operations that are supported by WS-Resources. The table lists the principle WSRF operations and provides a description of each one, including its message and response format.

Operation	Description

GetResourcePropertyDocument	Returns the entire resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:GetResourcePropertyDocument/>`
	Response format
	`<wsrf-rp:GetResourcePropertyDocumentResponse>`
	`  {any}`
	`</wsrf-rp:GetResourcePropertyDocumentResponse>`
	where *{any}* is the content of the resource properties document.

PutResourcePropertyDocument	Replaces the entire resource properties document for the WS-Resource with the document specified.
	Message format
	`<wsrf-rp:PutResourcePropertyDocument>`
	`  {any}`
	`</wsrf-rp:PutResourcePropertyDocument>`
	where *{any}* is the content of the new resource properties document.
	Response format
	`<wsrf-rp:PutResourcePropertyDocumentResponse>`
	`  {any} ?`
	`</wsrf-rp:PutResourcePropertyDocumentResponse>`
	where *{any}* is the content of the new resource properties document. If the content is the same as the requested content, then the *{any}* element must not be present.

GetResourceProperty	Returns the value or values of the specified resource property found within the resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:GetResourceProperty>`
	`  QName`
	`</wsrf-rp:GetResourceProperty>`
	Response format
	`<wsrf-rp:GetResourcePropertyResponse>`
	`  {any}*`
	`</wsrf-rp:GetResourcePropertyResponse>`
	where *{any}** is a sequence of elements that match the *QName* specified in the request.

GetMultipleResourceProperties	Returns the value or values of the specified resource properties found within the resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:GetMultipleResourceProperties>`
	`  <wsrf-rp:ResourceProperty>QName<wsrf-rp:ResourceProperty>+`
	`</wsrf-rp:GetMultipleResourceProperties>`
	Response format
	`<wsrf-rp:GetMultipleResourcePropertiesResponse>`
	`  {any}*`
	`</wsrf-rp:GetMultipleResourcePropertiesResponse>`
	where *{any}** is a sequence of elements that match the *QNames* specified in the request.

InsertResourceProperties	Inserts the resource property elements specified into the resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:InsertResourceProperties>`
	`  <wsrf-rp:Insert>`
	`    {any}*`
	`  </wsrf-rp:Insert>`
	`</wsrf-rp:InsertResourceProperties>`
	where `{any}*` is a sequence of elements with the same QName.
	Response format
	`<wsrf-rp:InsertResourcePropertiesResponse/>`

Table 67. Principle WSRF operations that are supported by WS-Resources (continued). The table lists the principle WSRF operations and provides a description of each one, including its message and response format.

Operation	**Description**

UpdateResourceProperties	Updates the resource property elements specified into the resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:UpateResourceProperties>`
	`  <wsrf-rp:Upate>`
	`    {any}*`
	`  </wsrf-rp:Upate>`
	`</wsrf-rp:UpateResourceProperties>`
	where *{any}** is a sequence of elements with the same QName.
	Response format
	`<wsrf-rp:UpateResourcePropertiesResponse/>`

DeleteResourceProperties	Deletes the resource property elements specified from the resource properties document for the WS-Resource.
	Message format
	`<wsrf-rp:DeleteResourceProperties>`
	`  <wsrf-rp:Delete ResourceProperty="QName"/>`
	`</wsrf-rp:DeleteResourceProperties>`
	where *QName* is the QName of the resource property to delete.
	Response format
	`<wsrf-rp:DeleteResourcePropertiesResponse/>`

QueryResourceProperties	Query the resource properties document by using a query expression, such as XPath.
	Message format
	`<wsrf-rp:QueryResourceProperties>`
	`  <wsrf-rp:QueryExpression`
	`        Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">`
	`    xsd:any`
	`  </wsrf-rp:QueryExpression>`
	`</wsrf-rp:QueryResourceProperties>`
	where *xsd:any* is the XPath query expression to apply to the resource properties document.
	Response format
	`<wsrf-rp:QueryResourcePropertiesResponse>`
	`  {any}`
	`</wsrf-rp:QueryResourcePropertiesResponse>`
	where *{any}* is the result of evaluating the query expression against the resource properties document.

Destroy	Destroys the WS-Resource.
	Message format
	`<wsrf-rl:Destroy/>`
	Response format
	`<wsrf-rl:DestroyResponse/>`
	This response indicates successful destruction of the WS-Resource.

Table 67. Principle WSRF operations that are supported by WS-Resources (continued). The table lists the principle WSRF operations and provides a description of each one, including its message and response format.

Operation	Description

SetTerminationTime	WS-Resources that support scheduled termination can implement this operation to allow a requester to change the time at which the WS-Resource destroys itself.
	Message format
	`<wsrf-rl:SetTerminationTime>`
	`  [<wsrf-rl:RequestedTerminationTime>`
	`    xsd:dateTime`
	`  </wsrf-rl:RequestedTerminationTime>]`
	`    |`
	`  [<wsrf-rl:RequestedLifetimeDuration>`
	`    xsd:duration`
	`  </wsrf-rl:RequestedLifetimeDuration>]`
	`</wsrf-rl:SetTerminationTime>`
	where the termination time is either an absolute time or a relative duration.
	Response format
	`<wsrf-rl:SetTerminationTimeResponse>`
	`  <wsrf-rl:NewTerminationTime>`
	`    xsd:dateTime`
	`  </wsrf-rl:NewTerminationTime>`
	`  <wsrf-rl:CurrentTime>`
	`    xsd:dateTime`
	`  </wsrf-rl:CurrentTime>`
	`<wsrf-rl:SetTerminationTimeResponse>`
	This response contains the time, from the perspective of the WS-Resource, when the WS-Resource destroys itself. The response also contains the WS-Resource value of the current time.
	A variety of ways exist in which a WS-Resource can implement scheduled destruction. For example, a WS-Resource that is implemented as an enterprise bean might use the enterprise bean container timer service by implementing the ejbTimeout callback method of the javax.ejb.TimedObject interface, and by creating a Timer object that expires at the scheduled destruction time and drives this callback method. EJB timer service Timer objects are retained after server restarts, and are therefore a simple means to manage the lifecycle of WS-Resources that have a finite lifecycle and require a time-based destruction mechanism.

Web Services Distributed Management

Web Services Distributed Management (WSDM) is an OASIS approved standard that supports managing resources through a standardized web service interface. Your environment, such as WebSphere Application Server host or an operating system host that has an exposed resource as a web service within a single interface is used to manage and control resources. WSDM is a distributed management model, but it does not replace any existing WebSphere Application Server administration models. WSDM provides a new way to expose the internal product administration functions for a web service interface.

The existing administration interfaces, such as managed bean (MBean), wsadmin, and Java Application Programming Interface (API), are more language and platform specific. WSDM provides a common, flexible infrastructure to manage the product resources by leveraging the web services protocols.

WSDM defines two specifications: Management Using Web Services (MUWS) and Management of Web Services (MOWS). MUWS defines how resources interact with the resources managed through a set of accessible web services interfaces. MOWS extends the MUWS concepts to define how a web service resource, itself, is managed. See Specifications and API documentation for MOWS and MUWS specifications. In addition to the manageability capabilities defined in the MUWS specifications, WebSphere Application Server WSDM also defines manageability capabilities unique to the product environment.

There is a general pattern that managed resources use to expose their manageability services through WSDM compliant web services interfaces. First, you must create a model of the managed resource. Typically the model of the resource is created using a modeling tool such as the Test and Performance

Tools Platform (TPTP), an eclipse plug-in tool; however, a simple text document is sufficient. Use the modeling tool to develop the model of WebSphere Application Server managed resources. The following graphic illustrates the process.

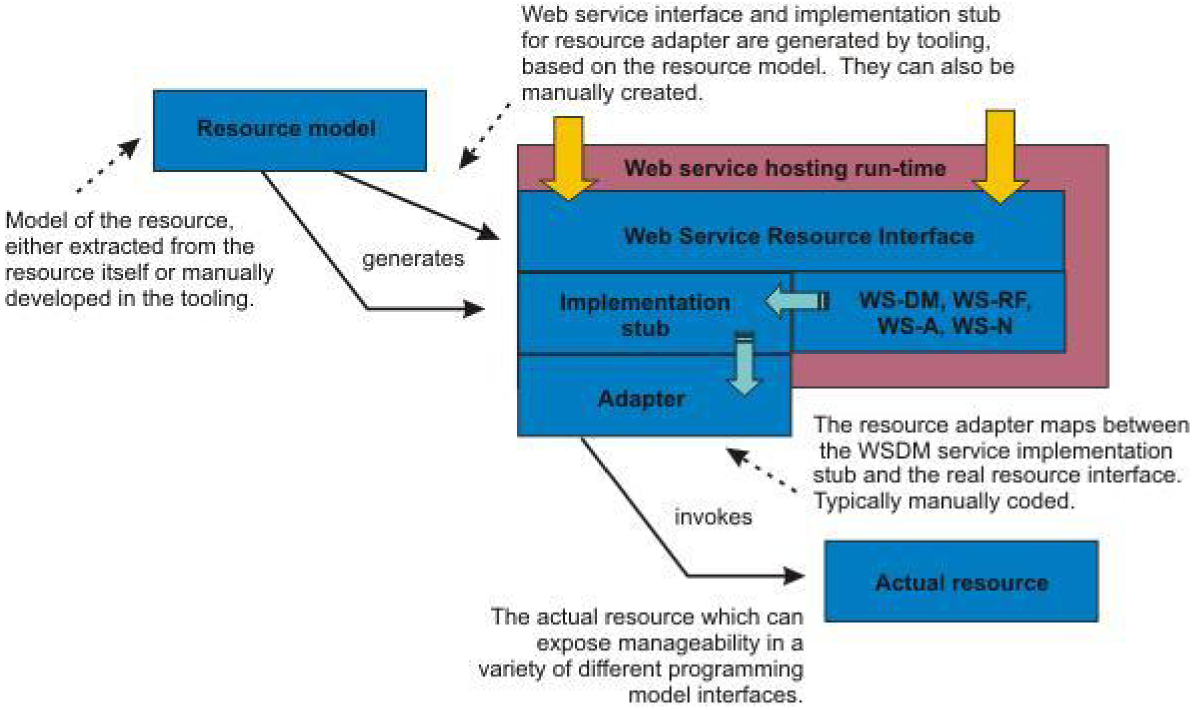

Web service interface and implementation stub for resource adapter are generated by tooling, based on the resource model. They can also be manually created.
Resource model
Model of the resource, either extracted from the resource itself or manually developed in the tooling.
generates
Web service hosting run-time
Web Service Resource Interface
Implementation stub
WS-DM, WS-RF, WS-A, WS-N
Adapter
The resource adapter maps between the WSDM service implementation stub and the real resource interface. Typically manually coded.
invokes
Actual resource
The actual resource which can expose manageability in a variety of different programming model interfaces.

Figure 109. Generic WSDM Concept

Code artifacts are generated from the resource model. Generated artifacts for each resource model include:

- A Web Services Description Language (WSDL) document that describes the web service interface for the management functions for that resource
- An implementation stub for the service implementation classes for that web service
- A client proxy for the service that is used in a program that needs to invoke the management functions of that resource
- A unit test code for invoking test cases that exercises the functions of that service
- Additional XML documents and schema that describe the properties, operations, and notifications associated with the managed resource

The code generated from the resource model is essentially an empty shell of the management web service for the modeled resource. The next step in the process is to enter code that acts as an adapter between the implementation stub for the service and the real resource management functions. In the case of the WSDM support implementation, this adapter code contain calls to theWebSphere Application Server AdminService APIs that expose normal product management functions. You must install the completed service implementation in a hosting web service environment. To install your WSDM application, see Deploying and administering enterprise applications and follow the steps for installing enterprise application files on an application server.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

Web Services Distributed Management resource management

Web Services Distributed Management (WSDM) is an OASIS approved standard that supports the management of resources through a standardized web service interface. WSDM delivers web services based interfaces to manage application server resources using a manageability endpoint.

The manageability endpoint contains manageability capabilities for the resources. A manageability capability uniquely identifies and associates with a set of properties, operations, and events. A resource that supports one or more manageability capabilities becomes a manageable resource. For example, a manageable resource is a server or an application resource that supports a capability, which includes stop, start, and remove operations. To leverage the functions that a manageable resource provides, a manageability consumer is used. Manageability consumer queries and discovers the available manageable resources through the web services endpoints. After the service is discovered, the manageability consumer exchanges messages to gather property information, invoke operations or receive notifications. The following graphics illustrates the relationships between the manageability consumer and manageable resources linked by the web service endpoints.

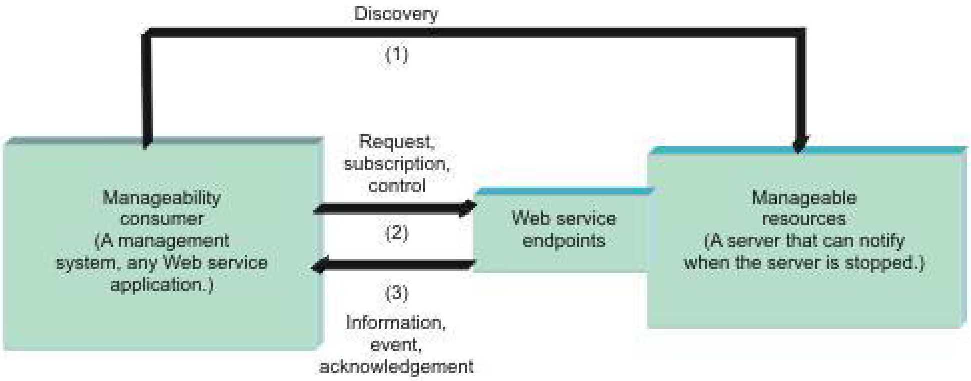


Figure 110. Relationship between the different parts of WSDM

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

Web Services Distributed Management manageability capabilities for WebSphere Application Server resource types

A resource that supports one or more manageability capabilities is a manageable resource. Each resource type that is exposed within the product supports a number of Web Services Distributed Management (WSDM) manageability capabilities.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

A manageable resource is a server or an application that supports a capability which includes stop, start, and remove operations. A manageability capability includes some properties, operations, and notifications. You can obtain and view performance data about the managed resources when you enable Performance Monitoring Infrastructure (PMI) in your server environment.

WSDM manageable resources, in general, are an aggregation of manageability capabilities. There are manageability capabilities that are globally applicable to many resource types. State management fits into this category. There are manageability capabilities that are unique to a single-managed resource, for example the Java virtual machine (JVM) manageability capability only applies to JVM-managed resources.

The autonomic manager (AC), which can be any client with management capability interact with the resources. Before the AC can interact with the resources, the AC needs to query what resources are available in the application server via the service group. The service group is an aggregation of WS-Resources within the same domain. The WebSphere Application Server WSDM service group contains all of the resources. Each resource becomes a member in the service group. The AC can get a particular resources endpoint reference (EPR) from the service group based on the resource type or the reference parameters. After the EPR is obtained, the AC can send the request to the resource. The service group can be accessed using the following endpoint address: http://<hostname>:<port>/websphere-management/services/service-group.

After the AC gets the resources EPR list from the service group, the AC can send requests to the resource provider. Each resource endpoint is listed below. The associated Web Services Description Language (WSDL) can be obtained by attaching `?wsdl` to the end of the endpoint address.

Resource type	Resource endpoint address
WebSphere Application Server profile, also called runtime configuration instance or WebSphere Application Server domain	http://<hostname>:<port>/websphere-management/services/webspheredomain
WebSphere Application Server	http://<hostname>:<port>/websphere-management/services/applicationserver
WebSphere Application Server cluster	http://<hostname>:<port>/websphere-management/services/webspherecluster
Java virtual machine	http://<hostname>:<port>/websphere-management/services/jvm
Application	http://<hostname>:<port>/websphere-management/services/application
WebSphere Application Server deployed object	http://<hostname>:<port>/websphere-management/services/deployedobject
Servlet	http://<hostname>:<port>/websphere-management/services/servlet
Enterprise JavaBeans	http://<hostname>:<port>/websphere-management/services/ejb
Web services	http://<hostname>:<port>/websphere-management/services/webservices
JAX-WS web services	http://<hostname>:<port>/websphere-management/services/jaxwswebservices
JAX-RPC web services	http://<hostname>:<port>/websphere-management/services/jaxrpcwebservices
Data source	http://<hostname>:<port>/websphere-management/services/datasource

Each resource type that is exposed in the product supports a number of manageability capabilities. These resources are defined by the WSDM specification, AC touchpoint, and the product's built-in management. A touchpoint is a combination of port types and operations defined in WSDL that exposes the manageability interface for a managed resource in a way that complies with different specifications for web services. Each manageability capability includes a number of properties, operations, and notifications.

The following table lists the manageability capabilities that each resource aggregates. For information about an Application Programming Interface (API) or a specification that is listed with a manageability capability, see Specifications and API documentation.

Resource types and manageability capabilities

Resource type	Manageability capabilities	Specification
WebSphere Application Server domain	- J2EEDomain - J2EEManagedObject - Identity - Metrics - ManageabilityCharacteristics - Description - ResourceType - Configuration - ApplicationManagement - ConfigChangeNotifier - NotificationProducer	- JSR 77 – J2EE - JSR 77 – J2EE - MUWS – WSDM - MUWS – WSDM - MUWS - MUWS – WSDM - AC touchpoint - MUWS – WSDM - WebSphere Application Server unique - WebSphere Application Server unique - WSBN - WS-N
WebSphere Application Server	- J2EEServer - J2EEManagedObject - Identity - Metrics - State - ManageabilityCharacteristics - Description - ResourceType - NotificationProducer - ApplicationServer - StateManageable	- JSR 77 – J2EE - JSR 77 – J2EE - MUWS – WSDM - MUWS – WSDM - MUWS – WSDM - MUWS - MUWS – WSDM - AC touchpoint - WSBN – WS-N - WebSphere Application Server unique - WebSphere Application Server unique
WebSphere Application Server cluster	- Identity - Metrics - State - ManageabilityCharacteristics - Description - ResourceType - ClusterManagement	- MUWS – WSDM - MUWS – WSDM - MUWS – WSDM - MUWS - MUWS – WSDM - AC touchpoint - WebSphere Application Server unique
Java virtual machine	- JVM - J2EEManagedObject - Identity - Metrics - ManageabilityCharacteristics - Description - ResourceType	- JSR 77 – J2EE - JSR 77 – J2EE - MUWS – WSDM - MUWS – WSDM - MUWS - MUWS – WSDM - AC touchpoint

Resource type	Manageability capabilities	Specification
Application	• J2EEApplication • J2EEDeployedObject • J2EEManagedObject • Identity • State • Metrics • ManageabilityCharacteristics • Description • ResourceType • Application • StateManageable	• JSR 77 – J2EE • JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS - WSDM • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • WebSphere Application Server unique
Servlet	• Servlet • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint
Enterprise JavaBeans	• EJB • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint
Web service	• Metrics • J2EEManagedObject • Identity • State • ManageabilityCharacteristics • Description • ResourceType • WebService	• MOWS – WSDM • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique

Resource type	Manageability capabilities	Specification
JAXWS web services	• J2EEManagedObject • Identification • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • WebService • Manageability references • OperationalStatus • Operational state • Operation operational status • Request processing state • Identity	• JSR 77 – J2EE • MOWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MUWS – WSDM
JAXRPC web services	• Metrics • J2EEManagedObject • Identification • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • WebService • Manageability references • OperationalStatus • Operational state • Operation operational status • Request processing state • Identity	• MOWS – WSDM • JSR 77 – J2EE • MOWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MUWS – WSDM
Data source	• JDBCDatasource • J2EEResource • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType • DataSource	• JSR 77 – J2EE • JSR 77 – J2EE • JSR 77 – J2EE • MUWS - WSDM • MUWS - WSDM • MUWS – WSDM • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique

The following table lists the attributes and operations for the product's manageability capabilities.

Manageability Capabilities	**Attributes**	**Operations**
J2EEDomain	None	• String getAttribute(String, String) • String[] queryNames(String queryString)
J2EEManagedObject	• objectName • stateMangeable • eventProvider • statisticsProvider	None
ConfigChangeNotifier	None	None (however, it has notification of ConfigChange)
ApplicationManagement	None	• String installApplication(String, String, HashMap) • String uninstallApplication(String) • String updateApplication(String, String, HashMap) • String, HashMap • EndpointReference listApplications(String applicationName)
J2EEServer	• serverVendor • serverVersion • DepolyedObjects • javaVMs	None
StateManageable	• state • startTime	• stop() • start() • startRecursive()
ApplicationServer	• name • versionsForAllEFixes • versionsForAllExtensions • VersionsForAllPTFs • shortName • threadMonitorInterval • threadMonitorthreshold • threadMonitorAdjustmentThreshold • ProcessId • cellName • nodeName • processType • platformName • platformVersion	• stopImmediate() • restart() • String getproductVersion(String)

Manageability Capabilities	Attributes	Operations
ClusterManagement	• clusterName • preferLocal • wlcId • state • backupName • backupBootstrapHost • backupBootstrapPort	• start() • stop() • stopImmediate() • rippleStarT() • exportRouteTable() • dumpClusterInfo() • boolean getAvailable(String, String) • boolean setAvailable(String, String) • boolean setUnavailable(String, String)
Java virtual machine	• javaVersion • javaVendor • node • stats • freeMemory • usedMemory • heapSize • upTime • GCCount • GCTime • GCInternalTime • waitsForLockCount • waitForLockTime • objectAllocatedCount • objectMovedCount • objectFreedCount • threadStartedCount • threadEndedCount	None
J2EEDeployedObject	• deploymentDescriptor • server	None
J2EE Application	module	None
Application	implementationVersion	None
Servlet	• concurrentRequest • responseTime • numErrors • totalRequests	None
EJB	• createCount • loadCount • storeCount • readyCount • liveCount • pooledCount • waitTime	None

Manageability Capabilities	Attributes	Operations
WebService	• payloadSize • replyPayloadSize • requestPayloadSize • requestResponseTime • replyResponseTime • responseTime • processRequestCount • dispatchedRequestCount • receivedRequestCount • loadedWebServiceCount	None
DataSource	• jdbcDriver • connectionFactoryType • dataSourceName • dataStoreHelperClass • loginTimeout • statementCacheSize • jtaEnabled • name • jndiName • testConnection • testConnectionInterval • stuckTimerTime • stuckTime • stuckThreshold • surgeThrehold • surgeCreationInterval • connectionTimeout • maxConnections • minConnctions • purgePolicy • reapTime • unusedTimeout • agedTimeout • freePoolDistributionTableSize • freePoolPartions • sharedPoolPartitions	• String showPoolContents() • void purgePoolContents() • void pause() • void resume() • String getStatus()

Web Services Distributed Management support in the application server

The Web Services Distributed Management (WSDM) support for a Web service in WebSphere Application Server runs within an application server that has exposed management functions.

In the application server implementation of WSDM, a WSDM application is packaged as a Java Platform, Enterprise Edition (Java EE) Enterprise archive (EAR) file. The EAR file is deployed as an application server system application.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

WSDM support for the product consists of two parts:

- WSDM runtime environment and support
- WSDM resource model and service implementation

The WSDM runtime environment provides fundamental capabilities for the manageable resources. The WSDM runtime environment interacts with the underlying web services platform and the WSDM resources to service the requests and responses. There are multiple specifications that the WSDM runtime environment uses in order to provide the WSDM functions, namely `WS-Addressing`, `WS-ResourceFramework`, and `WS-Notification`. For each request, the WSDM runtime environment routes the request to the appropriate resource service implementation based on the endpoint reference, (EPR). The EPR is defined by the `WS-Addressing` specification. Each EPR contains target address, runtime specific data and reference properties to uniquely identify an instance of a WSDM resource. After the resource service implementation returns a response, the WSDM runtime environment wraps the response into an appropriate SOAP message format specified in the Management Using Web Services (MUWS) specification and returns the response back to the requester. The application server leverages Apache Muse 2.0 to provide the runtime support for WSDM. The Apache MUSE 2.0 provides both the development tool and the WSDM runtime environment.

The WSDM resource model for the application server identifies the elements of the product that are managed resources and further defines the specific properties, operations, and notifications that are managed resources support. The resource model defines the interfaces to interact with the resources and administrative functions in the product. The resource model includes appropriate capabilities defined in the two WSDM specifications, Management Using Web Services (MUWS) and Management of Web Services (MOWS). What this means is that the implementation is a mapping of the WSDM specification interfaces onto the product administration and programming interfaces. The implementation does not introduce new functions into the product, but rather, an alternative interface for accessing existing administration and programming functions in the product. In addition, the resource model defines specific capabilities to provide additional manageability functions. Each of the capabilities defines a set of properties, operations, and events for managed resources in an autonomically managed system. Each resource is associated with a Web Services Description Language (WSDL) file that contains the definition of its manageability capabilities.

The implementation is attached to the WSSecurity default policy set and runs the administrative operations from the client user identity. This user identity must have privileges to perform any administrative action. It is the role of the autonomic computing (AC) manager that makes requests for the WSDM implementation to ensure that the user of that manager has appropriate authorization to perform administration and any other functions exposed by the AC manager.

The benefit of WSDM support in the application server is that the product can participate in multiple product management solutions in a standard way. By exposing the product management functions through a standard web services interoperable interface, you can combine the application server with large management systems based on the WSDM specification.

Web Services Distributed Management in a stand-alone application server instance

In a stand-alone application server environment, there is one Web Services Distributed Management (WSDM) application deployed for each application server instance.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

The WSDM application acts as an administrative client to the management code running inside the single Java virtual machine for that instance. Figure 1 illustrates an Autonomic Computing Manager (ACM) interacting with two application server instances, each with its own WebSphere Application Server WSDM application exposing the manageability for that individual instance.

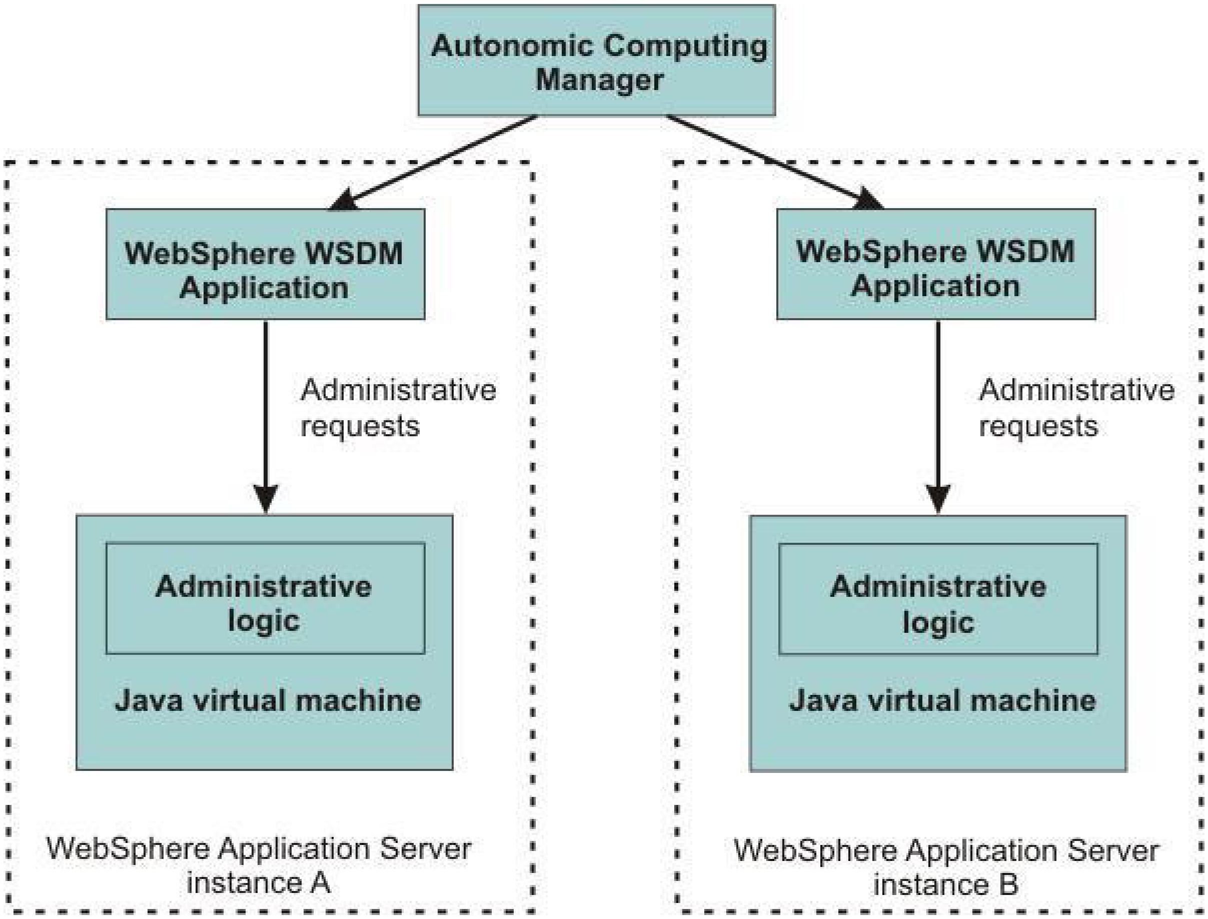

Autonomic Computing Manager
WebSphere WSDM Application
Administrative requests
Administrative logic
Java virtual machine
WebSphere Application Server instance A
WebSphere WSDM Application
Administrative requests
Administrative logic
Java virtual machine
WebSphere Application Server instance B

Figure 111. WSDM application in a stand-alone server instance

Web Services Distributed Management in a WebSphere Application Server, Network Deployment cell

You can use Web Services Distributed Management (WSDM) to manage application server instances within a WebSphere Application Server, Network Deployment cell. The administrative support and visibility for WSDM in a cell is obtained through interaction with each WSDM application deployed on the application server.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

In the multinode WebSphere Application Server, Network Deployment environment, the management code runs across a distributed network of Java virtual machines with a central access point as the deployment manager process for the entire network or cell. Several different application server Java virtual machines might be managed within a cell. You can manage an application server Java virtual machines within a cell through the WSDM application installed on deployment manager. The WSDM application acts as an administrative client to the managed application server. Figure 1 illustrates this environment with an

Autonomic Computing Manager interacting with the single application server implementation of WSDM to expose the manageability of that application server. You can build a federated deployment manager cell from individual application server instances by running the addNode utility program to add the application server instances to the centrally managed cell. After a node is added to the cell, the manager can still manage each application server within the cell through the installed WSDM application on the deployment manager.

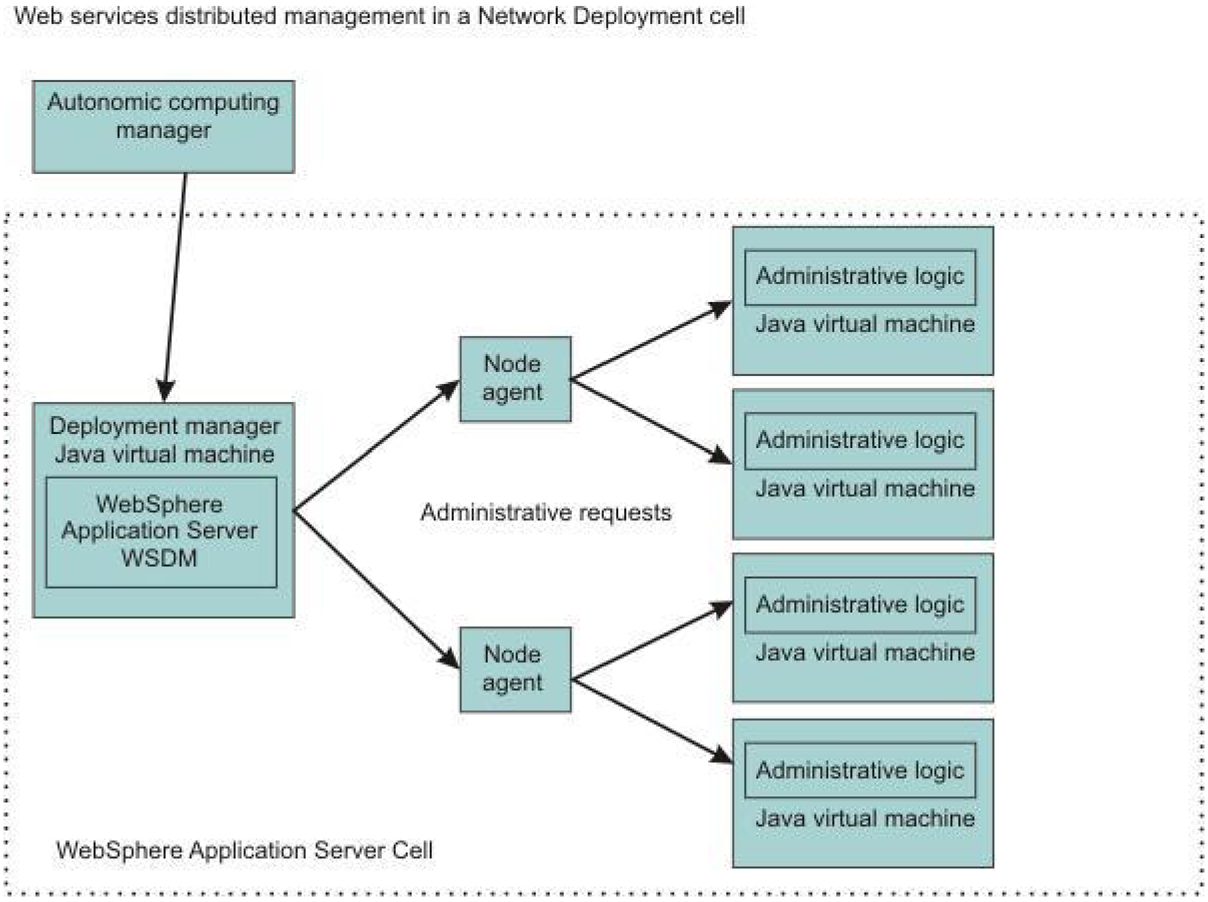


Figure 112. WSDM application in a WebSphere Application Server, Network Deployment cell

Web Services Distributed Management in an administrative agent environment

You can use Web Services Distributed Management (WSDM) to manage application server profiles in an administrative agent (AdminAgent) environment.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

You can manage multiple base application servers within profiles using the AdminAgent. You can also use the Web Services Distributed Management (WSDM) to manage each profile within the AdminAgent. WSDM is deployed inside the AdminAgent as a system application. Each profile is represented as a domain resource within the product. You can get and manage the resources within a profile through the product's domain resource. Figure 1 illustrates the AdminAgent topology.

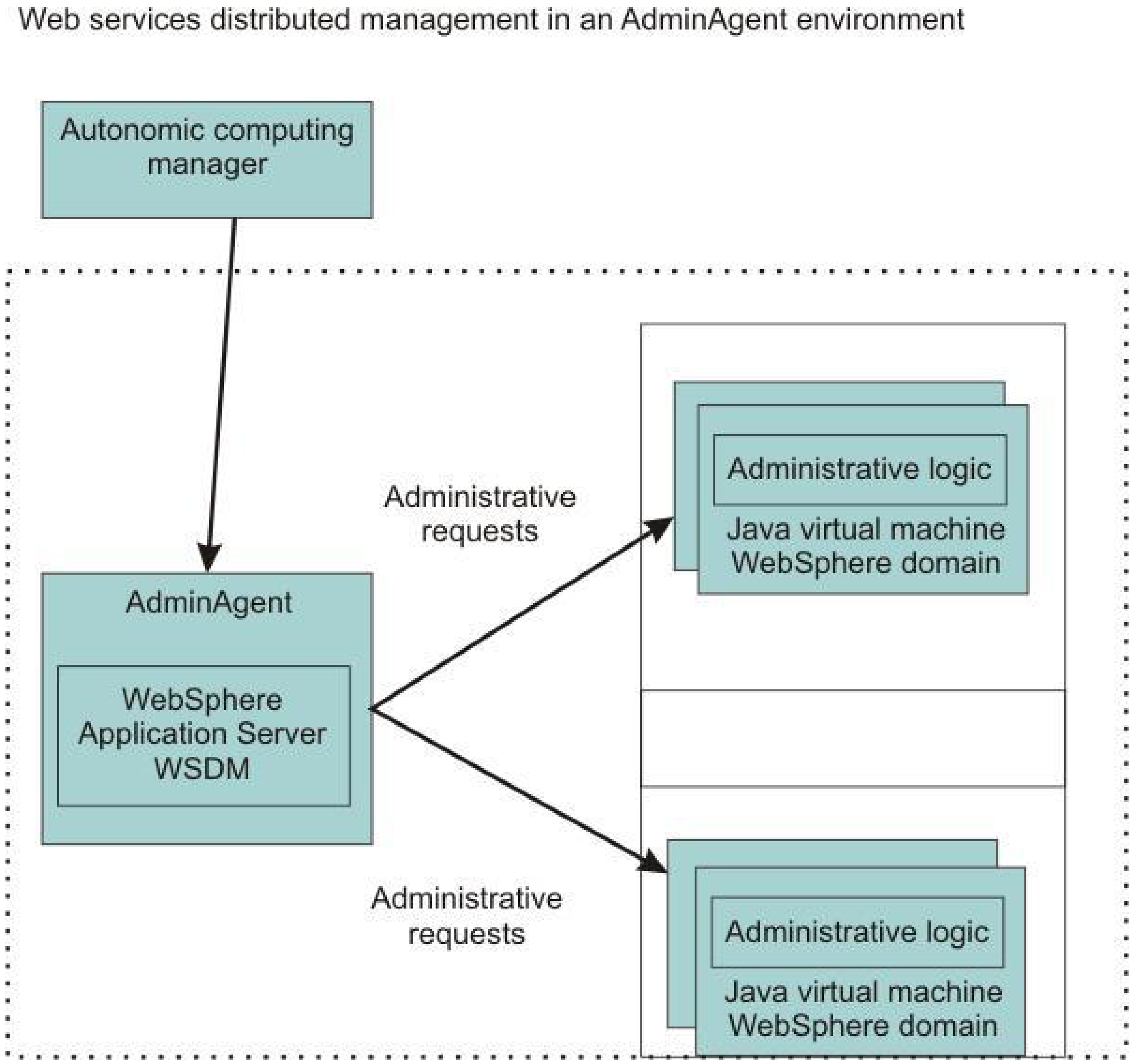

Web services distributed management in an AdminAgent environment
Autonomic computing manager
AdminAgent
WebSphere Application Server WSDM
Administrative requests
Administrative logic
Java virtual machine WebSphere domain
Administrative requests
Administrative logic
Java virtual machine WebSphere domain

Figure 113. WSDM application in an AdminAgent environment

Notifications from the application server Web Services Distributed Management resources

Use this topic to learn about application server Web Services Distributed Management (WSDM) resources and their life cycle events.

Important: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

There are different life cycle events that can occur for any resource. The notifications associated with these lifecycle events are resource definition events and resource state events. Resource definition events are:

- Created
- Deleted

- Changed

Resource state events are:
- Started
- Stopped
- Suspended

The following resources are discussed in detail. You can review WSDM manageability capabilities for application server resource types for information about each resource endpoint address and manageability capabilities.

The application server instances support definitional notifications. Whenever a resource definition event occurs, the configuration of an application server is created, modified, or deleted and a notification is generated by the WebSphere Application Server domain. The notification includes the configuration documents that have been changed.

The product installation is performed and managed by the underlying operating system. There is no runtime entity that represents the product installation. There are no life cycle notifications currently planned for events related to the product installation. There are no state event notifications associated with the lifecycle of the overall product installation.

There is an administrative agent process that is created as part of a WebSphere Application Server profile. This administrative agent process, once created, becomes available to emit life cycle notifications for the profile. Since any configuration modification might be considered a change to the profile, there is no generic *profile changed* notification. Instead, there are specific notifications for some configuration changes. In addition, there is no state change notification for profiles because a profile does not actually run, it simply exists or does not exist.

Applications that are installed into the product support both definitional and operational notifications. Whenever an application is installed, a notification is produced to indicate that an instance of that application managed resource has been created. A notification is generated each time the application is started, stopped, or updated. When an application is uninstalled, the *resource destroyed* notification is produced.

Individual deployed modules have independent life cycles. There are create, modify, delete, start, and stop notifications for individual deployed objects in the product.

Even though web services are not robust applications, there is a need to understand the life cycle for these essential deployed objects. Notifications are produced in accordance to the MOWS specification such as when a web service is installed, modified, started, stopped or uninstalled.

Web Services Invocation Framework (WSIF)

WSIF is a Web Services Description Language (WSDL)-oriented Java API. You use this API to invoke web services dynamically, regardless of the service implementation format (for example enterprise bean) or the service access mechanism (for example Java Message Service). Use these topics to learn more about WSIF.

Using WSIF, you can move away from the usual web services programming model of working directly with the SOAP APIs, towards a model where you interact with representations of the services. You can therefore work with the same programming model regardless of how the service is implemented and accessed.

Note: You should not use WSIF for new applications in WebSphere Application Server, unless you are supporting an existing WSIF configuration. You should instead adopt a more recent open standard, such as the Java API for XML-Based Web Services (JAX-WS) programming model.

To learn about WSIF, see the following topics:

- "Goals of WSIF."
- "WSIF Overview" on page 825.
  - "WSIF architecture" on page 826.
  - "WSIF and WSDL" on page 827.
  - "WSIF usage scenarios" on page 827.

Goals of WSIF

WSIF aims to extend the flexibility provided by SOAP services into a general model for invoking web services, irrespective of the underlying binding or access protocols.

SOAP bindings for web services are part of the Web Services Description Language (WSDL) specification, therefore when most developers think of using a web service, they immediately think of assembling a SOAP message and sending it across the network to the service endpoint, using a SOAP client API. For example: using Apache SOAP the client creates and populates a Call object that encapsulates the service endpoint, the identification of the SOAP operation to invoke, the parameters to send, and so on.

Although this process works for SOAP, it is limited in its use as a general model for invoking web services for the following reasons:

- "Web services are more than just SOAP services."
- "Tying client code to a particular protocol implementation is restricting" on page 825.
- "Incorporating new bindings into client code is hard" on page 825.
- "Multiple bindings can be used in flexible ways" on page 825.
- "A freer web services environment enables intermediaries" on page 825.

The goals of the Web Services Invocation Framework (WSIF) are therefore:

- To give a binding-independent mechanism for web service invocation.
- To free client code from the complexities of any particular protocol used to access a web service.
- To enable dynamic selection between multiple bindings to a Web service.
- To help the development of web service intermediaries.

Web services are more than just SOAP services

You can deploy as a web service any application that has a WSDL-based description of its functional aspects and access protocols. If you are using the Java Platform, Enterprise Edition (Java EE) environment, then the application is available over multiple transports and protocols.

For example, you can take a database-stored procedure, expose it as a stateless session bean, then deploy it into a SOAP router as a SOAP service. At each stage, the fundamental service is the same. All that changes is the access mechanism: from Java DataBase Connectivity (JDBC) to Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and then to SOAP.

The WSDL specification defines a SOAP binding for web services, but you can add binding extensions to the WSDL so that, for example, you can offer an enterprise bean as a web service that uses RMI-IIOP as the access protocol. You can even treat a single Java class as a web service, with in-thread Java method invocations as the access protocol. With this broader definition of a web service, you need a binding-independent mechanism for service invocation.

Tying client code to a particular protocol implementation is restricting

If your client code is tightly bound to a client library for a particular protocol implementation, it can become hard to maintain.

For example, if you move from Apache SOAP to Java Message Service (JMS) or enterprise bean, the process can take a lot of time and effort. To avoid these problems, you need a protocol implementation-independent mechanism for service invocation.

Incorporating new bindings into client code is hard

If you want to make an application that uses a custom protocol work as a web service, you can add extensibility elements to WSDL to define the new bindings. But achieving this capability is complex.

For example you must design the client APIs to use this protocol. If your application uses just the abstract interface of the web service, you must write tools to generate the stubs that enable an abstraction layer. These tasks can take a lot of time and effort. What you need is a service invocation mechanism that you can use to update existing bindings, and to add new bindings.

Multiple bindings can be used in flexible ways

To take advantage of web services that offer multiple bindings, you need a service invocation mechanism that can switch between the available service bindings at run time, without having to generate or recompile a stub.

Imagine that you have successfully deployed an application that uses a web service that offers multiple bindings. For example, imagine that you have a SOAP binding for the service and a local Java binding that lets you treat the local service implementation (a Java class) as a web service.

The local Java binding for the service can only be used if the client is deployed in the same environment as the service. In this case, it is more efficient to communicate with the service by making direct Java calls than by using the SOAP binding.

If your clients can switch the binding used based on runtime information, they can choose the most efficient available binding for each situation.

A freer web services environment enables intermediaries

Web services offer application integrators a loosely-coupled paradigm. In such environments, intermediaries can be very powerful.

Intermediaries are applications that intercept the messages that flow between a service requester and a target web service, and undertake some mediating task (for example logging, high-availability or transformation) before passing on the message. The Web Services Invocation Framework (WSIF) is designed to make building intermediaries both possible and simple. Using WSIF, intermediaries can add value to the service invocation without needing transport-specific programming.

WSIF Overview

The Web Services Invocation Framework (WSIF) provides a Java API for invoking web services, independent of the format of the service, or the transport protocol through which it is invoked.

WSIF provides the following features:

- An API that provides binding-independent access to any web service.
- A close relationship with Web Services Description Language (WSDL), so it can invoke any service that you can describe in WSDL.

- A stubless and completely dynamic invocation of a web service.
- The capability to plug a new or updated implementation of a binding into WSIF at run time.
- The option to defer the choice of a binding until run time.

WSIF provides runtime support for web services, and for WSDL extensions and bindings, that were not known at build time. This capability is known as *dynamic invocation*. Using WSIF, a client application can choose dynamically the optimal binding to use for invoking Web service operations. For example, a web service might offer a SOAP binding, and also a local Java binding so that you can treat the local service implementation (a Java class) as a web service. If a client application is deployed in the same environment as the service, this client can use the local Java binding for the service. This provides more efficient communication between the client and the service by making direct Java calls, rather than indirect calls that use the SOAP binding.

WSIF provides this runtime support through the use of providers that link the WSIF service to the underlying implementation of the service. The providers support web services, WSDL extensions, and bindings that were not known at build time by using the WSDL description to access the target service.

WSIF is designed to work both in an unmanaged environment (running WSIF as a client) and inside a managed container. You can use the Java Naming and Directory Interface (JNDI) to find the WSIF service, or you can use the location described in the WSDL.

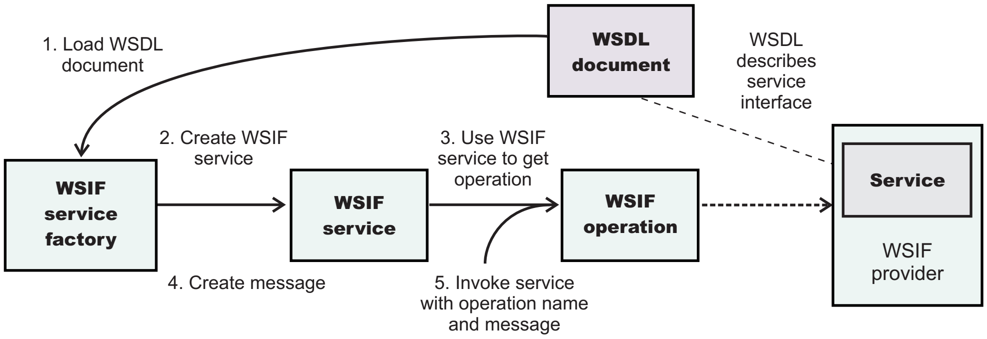
For more conceptual information about WSIF and WSDL, see the following topics:
- WSIF and WSDL
- WSIF architecture
- WSIF usage scenarios

WSIF supports Internet Protocol Version 6, and Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 for SOAP.

WSIF architecture

A diagram depicting the Web Services Invocation Framework (WSIF) architecture, and a description of each of the major components of the architecture.

The Web Services Invocation Framework (WSIF) architecture is shown in the figure.



The components of this architecture include:

WSDL document

The Web Services Description Language (WSDL) document contains the location of the web service. The binding document defines the protocol and format for operations and messages defined by a particular portType.

WSIF service

The WSIFService interface is responsible for generating an instance of the WSIFOperation interface to use for a particular invocation of a service operation.

WSIF operation

The runtime representation of an operation, called *WSIFOperation* is responsible for invoking a service based on a particular binding.

WSIF provider

A WSIF provider is an implementation of a WSDL binding that can run a WSDL operation through a binding-specific protocol. WSIF includes SOAP providers, JMS providers, Java providers and EJB providers. These providers link the WSIF service to the underlying implementation of the service.

WSIF and WSDL

There is a close relationship between the metadata-based Web Services Invocation Framework (WSIF) and the evolving semantics of Web Services Description Language (WSDL).

In WSDL, a service is defined in three distinct sections:

- The **portType**. This section defines the abstract interface offered by the service. A portType defines a set of *operations*. Each operation can be In-Out (request-response), In-Only, Out-Only and Out-In (Solicit-Response). Each operation defines the input and/or output *messages*. A message is defined as a set of *parts*, and each part has a schema-defined type.
- The **binding**. This section defines how to map between the abstract portType and a real service format and protocol. For example the SOAP binding defines the encoding style, the SOAPAction header, the namespace of the body (the targetURI), and so on.
- The **port**. This section defines the location (endpoint) of the available service. For example, the HTTP web address at which a SOAP service is available.

Currently in WSDL, each port has one and only one binding, and each binding has a single portType. But (more importantly) each service (portType) can have multiple ports, each of which represents an alternative location and binding for accessing that service.

The Web Services Invocation Framework (WSIF) follows the semantics of WSDL as much as possible:

- The WSIF dynamic invocation API directly exposes runtime equivalents of the model from WSDL. For example, invocation of an operation involves executing an operation with an input message.
- WSDL has extension points that support the addition of new ports and bindings. This enables WSDL to describe new systems. The equivalent concept in WSIF is a *provider*, which links the WSIF service to the underlying implementation of the service. This enables WSIF to understand a class of extensions and thereby to support a new service implementation type.

As a metadata-based invocation framework, WSIF follows the design of the metadata. As WSDL is extended, WSIF is updated to follow.

The primary type system of WSIF is XML schema. WSIF supports invocation using dynamic proxies, which in turn support Java type systems, but when you use the WSIFMessage interface to invoke a Web service through the WSIF API you must populate WSIFMessage objects with data based on the XML schema types as defined in the WSDL document. You should define your object types by a canonical and fixed mapping from schema types into the runtime environment.

WSIF usage scenarios

There are two main scenarios that illustrate the role WSIF plays in the emerging web services environment: Redevelopment and redeployment, and service flow composition.

Scenario: Redevelopment and redeployment

When you first implement a web service, you create a simple prototype. When you want to move a prototype web service into production, you often have to redevelop and redeploy it.

The Web Services Invocation Framework (WSIF) uses the same API calls irrespective of the underlying technologies, therefore if you use WSIF:

- You can reimplement and redeploy your services without changing the client code.
- You can use existing reliable and high-performance infrastructures such as Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and Java Message Service (JMS) without sacrificing the location-independence that the web service model offers.

Scenario: Service flow composition

A service flow typically invokes a web service, then passes the response from one web service to the next web service, perhaps performing some transformation in the middle.

There are two key aspects to this flow that WSIF provides:
- A representation of the service invocation based on the metadata in Web Services Description Language (WSDL).
- The ability to build invocations based solely on the portType, which can therefore be used in any implementation.

For example, imagine that you build a meta-service that uses a number of services to build a process. Initially, several of those services are simple Java bean prototypes that are written and exposed through SOAP, but you plan to reimplement some of them as EJB components, and to out-source others.

If you use SOAP, it ties up multiple threads for every onward invocation, because they pass through the web server and servlet engine and on to the SOAP router. If you use WSIF to call the beans directly, you get much better performance compared to SOAP and you do not lose access or location transparency. Using WSIF, you can replace the Java bean implementations with EJB implementations without changing the client code. To move some of the web services from local implementations to external SOAP services, you just update the WSDL.

WS-Policy

WS-Policy is an interoperability standard that is used to describe and communicate the policies of a web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction.

WebSphere Application Server conforms to the web services Policy Framework (WS-Policy) specification. You can use the WS-Policy protocol to exchange policies in standard format. A policy represents the capabilities and requirements of a web service, for example whether a message is secure and how to secure it, and whether a message is delivered reliably and how this is achieved. You can communicate the policy configuration to any other client, service registry, or service that supports the WS-Policy specification, including non-WebSphere Application Server products in a heterogeneous environment.

For a service provider, the policy configuration can be shared in published Web Services Description Language (WSDL), in WSDL that is obtained by a client by using an HTTP GET request, or by using the Web Services Metadata Exchange (WS-MetadataExchange) protocol. The WSDL is in the standard WS-PolicyAttachments format.

For a client, the client can obtain the policy of the service provider in the standard WS-PolicyAttachments format and use this information to establish a configuration that is acceptable to both the client and the service provider. In other words, the client can be configured dynamically, based on the policies supported by its service provider. The provider policy can be attached at the application or service or service reference level.

Note: The following features were introduced in the JAX-WS 2.2 specification, which WebSphere Application Server supports from Version 8:

- You can specify transport level security on client WSDL acquisition. You can attach a system policy set to either an HTTP GET request or a WS-MetadataExchange request when obtaining provider policy. See the “Configuring the client policy to use a service provider policy” topic for further information.
- You can specify a policy set and binding for a service reference that is different from the policy set attachment for the service. By default, service references inherit their policy set and WS-Policy configuration from their parent service, however, if desired, the policy set and WS-Policy configuration can be overwritten. See the “Using WS-Policy to exchange policies in a standard format” topic and its child topics for further details.
- You can enable and configure WS-Addressing support on a client or service provider by adding WS-Policy assertions into the WSDL document. WebSphere Application Server will now process WS-Addressing information held within the WS-Policy aspect of an application's WSDL document and use it in the configuration of that application. See the “Enabling Web Services Addressing support for JAX-WS applications using WS-Policy” topic for further information.
- You can publish policy configuration relating to WS-Addressing based on JSR109 deployment descriptors or JAX-WS 2.2 features or annotations, as well as information based on policy sets. This ensures that the policy information published matches the run time behavior of the service. See the “Web service providers and policy configuration sharing” topic for further information.

The WS-Policy assertion specifications that are supported in this version of WebSphere Application Server are:

- WS-Policy. See Web Services Policy 1.5
- WS-Addressing. See Web Services Addressing 1.0 - Metadata.
- WS-AtomicTransaction. See Web Services Atomic Transaction Version 1.0, Web Services Atomic Transaction Version 1.1 and Web Services Atomic Transaction Version 1.2.
- WS-ReliableMessaging. See Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1.
- WS-SecurityPolicy. See WS-SecurityPolicy 1.2.

For details of the WS-Policy domains that are supported, see the following topics:

- WS-Addressing policy settings
- WS-ReliableMessaging settings
- WS-Security policy settings
- WS-Transaction policy settings

Web service providers and policy configuration sharing

A WebSphere Application Server service provider can share its current policy configuration through its Web Service Description Language (WSDL). The policy configuration is in standard WSDL WS-PolicyAttachment format so that it can be shared with other clients, service registries, or services that support the Web Services Policy (WS-Policy) specification.

You can make the policy configuration of a Java API for XML-Based Web Services (JAX-WS) service endpoint available to share in the following ways:

- Include the policy configuration of the service provider in the WSDL. The WSDL is then available to publish, or to obtain by using an HTTP GET request.
- Enable the Web Services Metadata Exchange (WS-MetadataExchange) protocol so that the policy configuration of the service provider is included in the WSDL and is available to a WS-MetadataExchange GetMetadata request. An advantage of using the WS-MetadataExchange protocol is that you can apply message-level security to WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

System administrators can also access a WSDL document through a published compressed file with a `.zip` file extension, using the administrative console or administrative commands. However, a WSDL document acquired in this way might differ from a WSDL document acquired using an HTTP GET request or through the WS-MetadataExchange protocol, because the static WSDL document published in the compressed file will not have been able to take into account any web service features, annotations or deployment descriptor elements which may exist in the application code, such as WS-Addressing annotations.

By default, policy sharing is off. To include the policy configuration of the service provider in the WSDL, and specify how it is shared, you can use the administrative console or wsadmin commands.

When policy sharing is on, any WS-Policy attachments that were in the WSDL previously are removed. Note that policy configuration information becomes available in the WSDL to publish, but it is not available if you view the WSDL document directly from the administrative console or if you publish the WSDL remotely by using an administrative agent.

If the service provider application uses multipart WSDL, all the WSDL must be local to the web service application. For more information about multipart WSDL, see the topic about WSDL.

A service provider that is configured to use Security Assertion Markup Language (SAML) can share policy for use by a WebSphere Application Server client or a service registry. Note that the SAML tokens are published in a proprietary format.

Application developers can specify that a service provider shares its policy configuration, and how it is shared, by using Rational Application Developer tools when a web service is generated. For more information, see the Rational Application Developer documentation.

Transport policy information is not included in the policy configuration because transport policies such as HTTP, SSL, and JMS cannot be expressed in WS-PolicyAttachment format.

Bootstrap policy information, for example, the policy to access a WS-Trust service, can be included in the policy configuration if the bootstrap policy is expressed in standard, publishable WS-PolicyAttachment format.

You can configure a service provider to share its policy configuration at application or service level. The policy configuration that is represented by the policy sets attached to any earlier levels will also be shared. Policy sets that are attached at earlier levels override the policy set configuration attached at a later level.

Policy information can be defined in several ways. The following list is in descending order of precedence. For example, the deployment descriptor method overrides the use of annotations or features in the application code, but is itself overridden by the use of policy sets.

- Policy is defined by attaching a policy set to the application.
- Policy is defined by the use of deployment descriptor elements within a `port-component-ref` element.
- Policy is defined using annotations or features in the application code.
- Policy is defined using WS-Policy attachments in the WSDL document packaged with the application.

The following information lays out the rules governing how policy configuration is published:

- When policy sharing is enabled, the WS-Policy attachments in the WSDL describe the policy configuration of the service.
- When policy sharing is not enabled:
  - The WSDL that is returned by an HTTP GET request is the WSDL packaged with the application.

    Note: Such WSDL is returned unaltered and so may contain pre-existing WS-Policy attachments that do not match the configuration of the service.

– If there is no specific WSDL document associated with the service, then the server runtime generates a WSDL document automatically and associates it with the service. In this case the WSDL will contain no WS-Policy attachments unless an @Addressing annotation is present on the service implementation, in which case the @Addressing annotation configuration is expressed in WS-Policy attachments in the generated WSDL.

Troubleshooting policy configuration sharing

A service provider might not be able to share its policy configuration because the configuration cannot be expressed in the standard WS-PolicyAttachments format. One reason might be because multiple incompatible policies are defined for a particular attach point. Another reason might be because there is not enough binding information to generate the standard policy. Policy configuration might include bootstrap policy, for example, the policy to access a WS-Trust service, so the bootstrap policy must also be expressed in WS-PolicyAttachments format.

If the policy configuration cannot be shared, an error that describes the problem is written to the service provider error log, and the following policy is attached to the WSDL of the service provider:

```
<wsp:Policy>
<wsp:ExactlyOne>
</wsp:ExactlyOne>
</wsp:Policy>
```

This policy notifies the client that there is no acceptable policy configuration for the service. Other aspects of the WSDL are unaffected.

Web service clients and policy configuration to use the service provider policy

If a service provider publishes its policy in its Web Services Description Language (WSDL), the policy configuration of a WebSphere Application Server service client can be configured dynamically, based on the policies supported by its service provider.

The service provider must publish its policy in WS-PolicyAttachment format in its WSDL and the client must be able to support those provider policies. The client can base its policy configuration entirely on the policy of the provider, or partly on the policy of the provider with restrictions that are defined by the policy set configuration of the client.

A client acquires the provider policy by using either an HTTP GET request or the Web Services Metadata Exchange (WS-MetadataExchange) protocol to obtain the WSDL of the provider. You can configure how the client obtains the provider policy, and the endpoint at which the policy is acquired, by using the administrative console or wsadmin commands. If you use the WS-MetadataExchange protocol to obtain the policy of the provider, this has the advantage that you can secure WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

If the provider policy uses multipart WSDL, you can use an HTTP GET request to obtain the policy of the provider, but you cannot use the WS-MetadataExchange protocol. For more information about multipart WSDL, see the topic about WSDL.

The web application client-side policy is calculated and cached as a runtime configuration. This calculated policy is known as the effective policy and is used for subsequent outbound web service requests to the endpoint or operation for which the calculation was performed. The original policy set configuration of the client does not change.

For a specific service, dynamic policy configuration occurs once by default, and it is assumed that this configuration is the same for all endpoints that implement a service, because they have the same WSDL. The policy calculations that are based on this WSDL are cached in the client runtime (they are not persisted) and shared with each target service.

In WebSphere Application Server Version 8.0 and later, a service reference can be configured to use a different WSDL document to the WSDL configured for the client service. By default, service references inherit their policy set and WS-Policy configuration from their parent service, however, if desired, the policy set and WS-Policy configuration can be overwritten. See Using WS-Policy to exchange policies in a standard format for further details.

If you require a different policy configuration for each endpoint implementation, you must create a new port for each endpoint. Then you can specify a different policy configuration for each endpoint.

Transport policies such as HTTP, SSL, and JMS, cannot be expressed in WS-PolicyAttachment format, so the client cannot acquire the transport policies of the service provider. If the client requires transport policies, you must configure these policies as part of the policy set configuration of the client.

For an HTTP GET request, when the request is targeted at the same location as the endpoint, the request uses the same HTTP and SSL transport policies as the application. When the HTTP GET request is targeted at a different endpoint, you can also attach a system policy set to specify different HTTP and SSL transport policies.

For a WS-MetadataExchange GetMetadata request, the WS-Security configuration in the specified system policy set is used. The HTTP transport properties are inherited from the application.

A client that is configured to use Security Assertion Markup Language (SAML) can use dynamic policy configuration. However, the client must be configured to use general bindings.

Policy in a registry

A client can obtain the policy configuration of a web service provider from a registry, such as WebSphere Service Registry and Repository (WSRR), by using an HTTP GET request.

The WSDL for the policy of the service provider, and its corresponding policies and policy attachments, are stored in a registry such as WSRR. That policy must contain its policy configuration in WS-PolicyAttachments format. The client must be able to support those provider policies.

The registry must support the use of HTTP GET requests to publish WSDL that contains WS-Policy attachments, for example WSRR Version 6.2 or later.

You can apply the provider policy that the client obtains from a registry at the service or service reference level, but not at the application level.

If there is a secure connection between the client and the registry, you must ensure that trust is established between the application server and the registry server.

If the registry requires authentication, you also have to configure a policy to authenticate outbound service requests to the registry. By default, the HTTP and HTTPS credentials are used for both the web service endpoint and the registry. Therefore, it is advisable to secure any authorization credentials and ensure that these credentials are not sent to an unauthorized endpoint. You can also attach a system policy set to specify different HTTP and SSL transport policies.

Policy inheritance

The provider policy can be attached at the application or service level. Endpoints and operations inherit their policy configuration from the relevant service.

Calculating policy

Policy intersection is the comparison of a client policy and a provider policy to determine whether they are compatible, and the calculation of a new policy that complies with both their requirements and capabilities. When you obtain the policy of a service provider, you can choose to use the provider policy only, or to use the client and the provider policy. The outcome of policy intersection is as follows:

- When you specify provider policy only, the calculated policy is based on all the policies that the WebSphere Application Server client supports intersected by the provider policy. Effectively, the provider determines the policy, as long as the client can support that policy. This policy configuration is available if the scope point (endpoint operation) where the provider policy is attached is not attached to a client policy set and does not inherit a policy set attachment from parent scope points.
- When you specify client and provider policy, the calculated policy is based on the policy that is acceptable to the client intersected by the provider policy. Effectively, the policy conforms to the client policy set, but might be restricted further by the policies dictated by the provider. The policy that is acceptable to the client is defined by the policy set that is either attached to the client scope point, or that the client scope point inherits from a parent scope point. This policy configuration is available if the scope point (endpoint operation) where the provider policy is attached is attached to a client policy set or inherits a policy set attachment from parent scope points.

The WS-Policy language provides a way to express multiple policy choices, so the policy calculation might produce more than one result. For example, the service provider might support both WS-ReliableMessaging 1.0 and WS-ReliableMessaging 1.1. If the client also supports both versions, the client can use either version in its web service requests to the provider. In this situation, where more than one specification version is acceptable to both the client and the provider, the effective policy is calculated by using the most recent version.

Policy intersection in the JAX-WS dispatch client

Invocations that use the JAX-WS dispatch client (javax.xml.ws.Dispatch) use provider policy in their configuration if this is the administered behavior for the service. If the operation for the invocation is unknown, the client behaves as follows:

- The client complies with the provider policy scoped to the operation only if the provider policy is identical for all the operations provided by the service (both semantically and syntactically).
- If the provider policy is not identical for all the operations provided by the service, the client returns a JAX-WS WebserviceException with the cause WSPolicyException (CWPOL0106E), and an appropriate error message.
- If there is no policy on any of the operations, the client uses the effective provider policy for the service endpoint.

Refreshing the provider policy held by the client

The provider policy that the client holds for a service is refreshed the first time that the web service is invoked after the application is started. After that, the provider policy is refreshed when the application restarts, or when you explicitly invoke an update of the provider policy. When the provider policy is refreshed, the effective policy is recalculated.

You can invoke an update of the provider policy in the application code. This might be useful if a JAX-WS invocation fails; in the exception handling, you can force a retry with refreshed policy. You can set the following property (available in the WSPConstants class of the API) on the JAX-WS client proxy, then reissue the JAX-WS request: com.ibm.websphere.wspolicy.refreshProviderPolicy.

When the com.ibm.websphere.wspolicy.refreshProviderPolicy property is set, the provider policy that the client holds for a service is refreshed, and the effective policy is recalculated at the next request. After the refresh and recalculation have occurred, the com.ibm.websphere.wspolicy.refreshProviderPolicy property is unset.

The following example of code for a dispatch client shows the identification of an exception that might be resolved by refreshing the provider policy, followed by the invocation of the refresh.

```
try
{
dispatch.invoke(params);
}
catch (javax.xml.ws.WebServiceException e)
{
Throwable cause = e.getCause();
if ((cause instanceof NullPolicyException) || (cause instanceof PolicyException) )
{
// The exception might be because the policy of the provider is not up to date.
//
// There is also a message on the console that starts with the characters CWPOL,
// which helps to decipher and debug the cause of the error.
// This message is also available by using
// String nlsedMessage = cause.getMessage();
Map<String, Object> requestContext = dispatch.getRequestContext();
requestContext.put(WSPConstants.REFRESH_PROVIDER_POLICY, Boolean.TRUE);
// The following method might cause another jax-ws invocation exception.
// The cause might still be policy, in which case, a message is written to the
// console.
dispatch.invoke(params);
}
// For all other exceptions, use the normal exception handling for the
// application. In this case, assume there are no other exceptions and rethrow the
// initial exception. Remember that the WebServiceException might be caused by a
// WSPolicyAdministrationException. In this situation, a message is written to the
// console, but forcing a refresh in the application cannot resolve the problem.
throw e;
}
```

WS-MetadataExchange requests

You can use the Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request to exchange Web Services Description Language (WSDL) that is annotated with WS-Policy information. A service provider can use a WS-MetadataExchange request to share its policies, and a service client can use a WS-MetadataExchange request to apply the policies of a provider. You can secure WS-MetadataExchange requests by using transport-level or message-level security.

The WS-MetadataExchange specification defines a mechanism to retrieve metadata from an endpoint. WebSphere Application Server supports the use of the WS-MetadataExchange 1.1 GetMetadata request to return metadata in a response. A service provider can use this mechanism to make WSDL that is annotated with WS-Policy information available, that is, the service provider can share its policies. A service client can use this mechanism to obtain WSDL that is annotated with WS-Policy information from a service provider and then apply those policies. The policy configuration must be in WS-PolicyAttachments format in the WSDL of the service provider.

You can use a WS-MetadataExchange request as an alternative to using an HTTP GET request.

By default, a service provider or a service client does not use WS-MetadataExchange to share or obtain WS-Policy information. You must configure the service provider to share its policies, or configure the service client to apply the policies of a service provider, and specify that a WS-MetadataExchange request is used to share or obtain the policy configuration. WS-Policy information can be shared or obtained at the application or service level. You can configure the service provider or service client by using the administrative console or by using wsadmin commands.

Application developers can configure the service provider or service client using Rational Application Developer tools when a Web service is generated. For more information, see the Rational Application Developer documentation.

When a service provider is configured to share its policies through WS-MetadataExchange, the service supports incoming WS-MetadataExchange GetMetadata requests that are limited to the WSDL dialect. When the service receives such a request, the WSDL of the service is returned inline through a conformant WS-MetadataExchange response. The WSDL of the service contains WS-PolicyAttachments annotations that represent the current policy configuration. The policy configuration is in WS-PolicyAttachments format in the WSDL so that it is then available to other clients, service registries or services that support the Web Services Policy (WS-Policy) specification and the WS-MetadataExchange GetMetadata request.

When a service client is configured to use WS-MetadataExchange to obtain the policy of a service provider, the service client sends a WS-MetadataExchange GetMetadata request that specifies the WSDL dialect whenever it needs to obtain or refresh the policy of the provider.

WS-MetadataExchange security

You must ensure that the GetMetadata request is secured so that there is effective authentication, authorization, integrity, and confidentiality. End-to-end authentication is particularly important for the exchange of security metadata (SecurityPolicy), because if an unauthorized party could access this information, security credentials could be sent to non-trusted endpoints.

The GetMetadata request is targeted at the same port as the application endpoint, so if the application uses transport-level security, the GetMetadata request is also be targeted at the secure port and will, by default, use the same transport-level security configuration of the application.

Additionally, you can apply message-level security (WS-Security) to the metadata exchange. You might want to apply message-level security if transport-level security is not available on the application endpoint, or if transport-level security is not adequate for your requirements. An advantage of message-level security is that it provides end-to-end security by incorporating security features in the header of the SOAP message.

To provide message-level security, you attach system policy sets and general (named) bindings to the endpoint when you configure the service provider or service client to exchange policy configurations.

System policy sets are used for system messages that are not business-related, whereas application policy sets specify policy assertions for business-related messages. For example, system policy sets are used for messages that apply qualities of service (QoS), which includes the messages that are defined in the WS-MetadataExchange protocol. To provide message-level security for a GetMetadata request, you must attach a system policy set that contains only Web Services Security (WS-Security) or Web Services Addressing (WS-Addressing) policies. You can specify general bindings that are scoped either to the global domain or to the security domain of the service.

When you apply message-level security, any transport policy of the application is always used.

WS-ReliableMessaging

WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. Use these topics to learn more about WS-ReliableMessaging.

Without WS-ReliableMessaging, your web services that require assured delivery of SOAP messages can either use a vendor-specific binding such as SOAP over JMS (which provides limited interoperability) or they can use SOAP over HTTP and rely upon you to write the associated durable message stores, custom retry logic at the sender, and duplicate detection at the receiver. With WS-ReliableMessaging, you can make your SOAP over HTTP-based web services reliable without having to write custom code.

To enable WS-ReliableMessaging for an application, you take the following broad actions:

1. Develop a Java API for XML-Based Web Services (JAX-WS) web service provider or requester application.
2. Install the application into WebSphere Application Server.

3. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
4. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

At any stage - that is, before or after you have built your reliable web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

To learn about the WS-ReliableMessaging implementation in WebSphere Application Server, see the following topics:

- “WS-ReliableMessaging - How it works”
- “Benefits of using WS-ReliableMessaging” on page 837
- “Qualities of service for WS-ReliableMessaging” on page 837
- “Use patterns for WS-ReliableMessaging” on page 839
- “WS-ReliableMessaging sequences” on page 842
- “WS-ReliableMessaging - terminology” on page 843
- “WS-ReliableMessaging: supported specifications and standards” on page 844

WS-ReliableMessaging - How it works

WebSphere Application Server uses WS-ReliableMessaging as part of the transport layer for SOAP over HTTP messages. The message exchange patterns supported at the API layer are one-way “fire and forget,” or two-way request and reply.

The reliability is provided by reliable messaging middleware that sits between the web service requester and the web service provider. This middleware layer is shown beneath the dotted line in the following diagram, and includes the reliable messaging source and the reliable messaging destination.

Note: When using WS-ReliableMessaging with a two-way programming API, if the requesting application fails and is restarted it will not receive its reply message. In this model, WS-ReliableMessaging is being used to protect from network failures only. Moreover:

- Client-side retransmissions only start after the client starts sending new messages to the service (this is the situation for both one-way and two-way operations).
- Two-way operations that resume cannot drive the response message back to the client application; the message only gets back as far as the inbound sequence on the client.

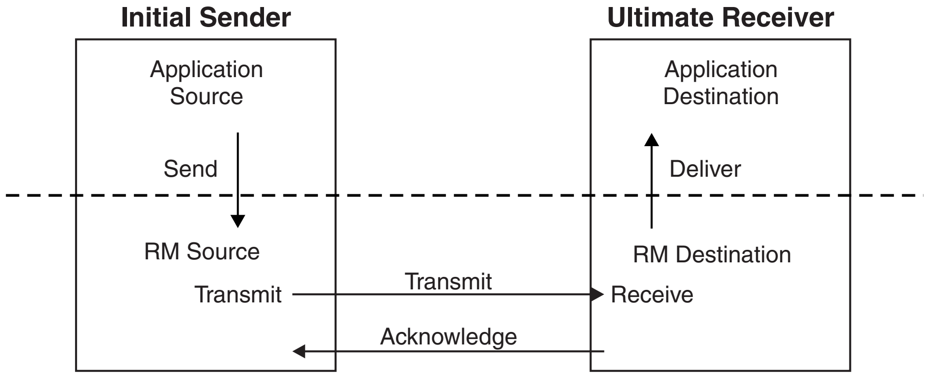


Figure 114. The interactions used to exchange web services messages reliably.

In the previous diagram, the application source invokes a web service. The sequence of interactions is as follows:

- The invocation is passed to the reliable messaging source.
- The reliable messaging source stores the message and then returns control to the application.
- The reliable messaging source sends the message to the reliable messaging destination.
- After the reliable messaging destination receives the message, it stores it locally and sends an acknowledgement message back to the reliable messaging source.
- The reliable messaging source can now delete its copy of the message.
- The reliable messaging destination can deliver the message to the application destination at any time after it receives it from the reliable messaging source.

To configure a web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent.

The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus. For each attachment between an application and a policy set, you can select the bus and messaging engine to use for the reliable messaging protocol state.

Benefits of using WS-ReliableMessaging

With WS-ReliableMessaging, along with the other components of the Reliable Secure Profile, you can support your business-to-business web services scenarios without having to write your own custom retry logic, duplicate detection code and persistence code.

WS-ReliableMessaging composes with other web services standards as described in “WS-ReliableMessaging: supported specifications and standards” on page 844.

Qualities of service for WS-ReliableMessaging

You can get different qualities of service with WS-ReliableMessaging, depending on the level of durability and transaction support provided by the store used to manage the reliable messaging state. These qualities of service range from protecting against loss of messages across a network, through to protecting against server failure.

WebSphere Application Server provides the following three qualities of service for WS-ReliableMessaging when using a SOAP over HTTP binding. All three qualities of service are supported when applications are deployed to the application server. Thin client and client container applications use the first option only.

Unmanaged non-persistent

You can configure web service applications to use WS-ReliableMessaging with a default in-memory store. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages.

Managed non-persistent

This in-memory quality of service option uses a messaging engine to manage the sequence state, and messages are written to disk if memory is low. This quality of service allows for the re-sending of messages that are lost in the network, and can also recover from server failure. However, state is discarded after a messaging engine restart so in this case you will lose messages.

Managed persistent

This quality of service for asynchronous web service invocations is recoverable. This option also uses a messaging engine and message store to manage the sequence state. Messages are persisted at the web service requester server and at the web service provider server, and are recoverable if the server becomes unavailable. Messages that have not been successfully transmitted when a server becomes unavailable can continue to be transmitted after the server restarts.

Note:

- The quality of service you get when using WS-ReliableMessaging is a direct result of the durability of the store managing the messages.
- When you use in-order delivery and either of the managed qualities of service, if the service causes an error then the message is re-dispatched to the service.
- You must ensure that when interacting with other vendors implementations of WS-ReliableMessaging, the other implementations provide the quality of service you require.

How the different qualities of service are implemented

When the web service application invokes the web service, the SOAP message is added into the WS-ReliableMessaging store. For the Managed qualities of service, the sending application transaction is used to put the message into the message store. After the transaction commits, the message is eligible for delivery. The other quality of service option is not transactional, so it considers the message eligible for delivery immediately.

The WS-ReliableMessaging protocol is used to reliably deliver the message to the target server where it is stored and acknowledged.

The message is read from the store and dispatched to the receiving application. For the Managed Persistent quality of service, a transaction is used to read the message and then dispatch the application.

For more information about using WS-ReliableMessaging transactions, see Providing transactional recoverable messaging through WS-ReliableMessaging.

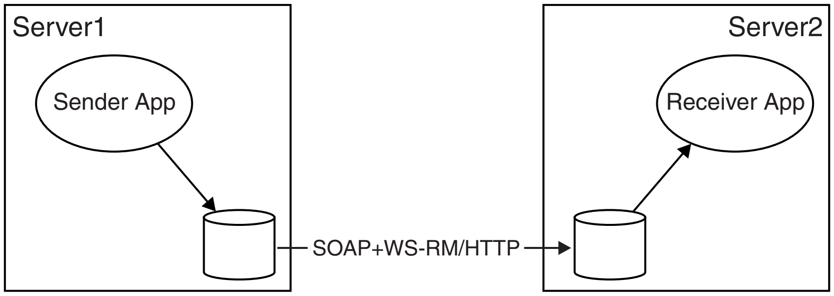


Figure 115. Using stores to exchange web services messages reliably.

The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus. For each attachment between an application and a policy set, you can select the bus and messaging engine to use for the reliable messaging protocol state.

Use patterns for WS-ReliableMessaging

Links to descriptions of the use patterns that motivate WS-ReliableMessaging. Each use pattern description includes an overview of the business problem, of the technical solution without WS-ReliableMessaging, of the shortcomings of this solution, and of how you can use WS-ReliableMessaging to overcome these shortcomings.

Historically, most business-to-business integration has been implemented on a point-to-point basis. However this situation is rapidly changing and hub-and-spoke is becoming more important, particularly for supply chain use patterns. The point-to-point use pattern is also important because, although your eventual goal might be to implement a business-to-business hub, you might nonetheless begin with point-to-point prototypes and proofs of concept.

For a description of each use pattern, see the following topics:

- “Assured delivery for B2B web services: point-to-point use pattern.”
- “Assured delivery for B2B web services: hub-and-spoke use pattern” on page 841.
- “Interoperation with other WS-ReliableMessaging providers: use pattern” on page 842.

Assured delivery for B2B web services: point-to-point use pattern

In this use pattern, a manufacturer sells its products through a network of affiliated dealerships. This manufacturer has initiated a pilot project to improve the IT integration between its own retail organization and half a dozen of the largest, most important dealerships.

The existing technical solution

Historically, business-to-business "e-commerce" has been conducted by using Electronic Data Interchange (EDI). EDI is a set of standards for the content and formatting of business-to-business messages. For examples of these standards and messages, see the United Nations Directories for Electronic Data Interchange.

If the identities of communication partners are known and unchanging, the use of industry standard message definitions is not strictly necessary. Although other XML-based standards are available for conducting business-to-business e-commerce (such as the OASIS Electronic Business using eXtensible Markup Language (ebXML) specifications) the manufacturer has decided to investigate the use of web services technologies, and is using WSDL documents from a variety of sources to define the service interfaces.

The interactions between the manufacturer and its dealers for the initial pilot project fall into two categories:

- Requests for information. The interaction is two-way, in that a request message is sent requesting some information, and a reply message is sent in the reverse direction containing the requested information. An example of a request for information going from a dealer to the manufacturer might be "getOrderStatus".
- Requests for update. These interactions are one-way, in that the sender of a request for update is not dependent on receiving a response in order to proceed with other work. An example of a request for update going from dealer to manufacturer might be "placeOrder". An example of a request for update going from manufacturer to dealer might be "deliveryConfirmed".

The manufacturer uses WebSphere Application Server to implement requests for information by using SOAP over HTTP and SOAP over JMS. Dealers are free to choose their own implementation technology; they do not have to use WebSphere Application Server.

The manufacturer implements requests for update in two different ways:

- Using SOAP over HTTP. In this case the service is represented as a request and reply interaction that is considered to have succeeded when the requestor successfully receives a reply. The services must be implemented to detect and successfully respond to duplicate requests (this is termed an idempotent operation), and the client has to be implemented to try again if the communication is interrupted after the request has been sent but before the reply has been received.
- To avoid the previous limitations, the manufacturer also uses SOAP over JMS support from WebSphere Application Server and WebSphere MQ. In this case the request is represented as a one-way service, and the messages are delivered reliably. The manufacturer uses WebSphere MQ as the JMS Provider, and makes this solution available to all dealers that also use WebSphere Application Server and WebSphere MQ. It is not required that the dealer and manufacturer be connected in order for the message to be sent.

The messages are transmitted over Virtual Private Networks, to ensure the integrity and confidentiality of messages transmitted between the two businesses, and as a part of establishing the identity of the sender.

The business problem

Although both the manufacturer and its dealers are happy with the implementation of the request for information services, there are a number of issues in the request for update case:

- Using SOAP over HTTP:
  - For the manufacturer, implementing idempotent services is complicated and therefore more expensive in developer time. It increases the likelihood of coding errors, reducing the robustness of the solution and introducing the possibility of expensive dropped or duplicated orders.
  - For dealers, implementing the retry logic is similarly complex, expensive, and error-prone.
  - For both the manufacturer and the dealers, the requirement for both to be available in order to invoke the service is an issue. In particular, many dealers do not maintain seven-day availability of their systems, whereas for the manufacturer weekends are the ideal time to deliver price updates to the dealers. Similarly, being unable to place orders when connectivity between dealer and manufacturer is unavailable is a real business issue.
- Using SOAP over JMS:
  - Although requiring the use of WebSphere Application Server and WebSphere MQ is acceptable to the current collection of dealers, as the project expands there might be other partners who are unwilling or unable to use a common software platform.

The solution when using WS-ReliableMessaging

With WS-ReliableMessaging support in WebSphere Application Server, the manufacturer can replace their existing custom-retry solutions for reliable one-way messaging with standard SOAP over HTTP one-way messaging. The removal of the retry logic from the application simplifies the application code, enabling simpler and quicker application development.

With WS-ReliableMessaging, the dealer and manufacturer do not have to be connected in order for the message to be sent.

The WS-ReliableMessaging standard adds reliability to SOAP over HTTP messaging, reducing the need to use SOAP over JMS.

Because WS-ReliableMessaging with SOAP over HTTP is an interoperable standard, the network of dealers need not use a common software platform.

Assured delivery for B2B web services: hub-and-spoke use pattern

In this use pattern, a manufacturer is looking for more than the ability to conduct transactions electronically with a fixed set of partners; the manufacturer needs a service that provides visibility to their inventory levels, so that the suppliers can manage their own product and inventory levels accordingly.

The manufacturer has many suppliers - around 2000 major suppliers and 250,000 other suppliers - and the set of supplier companies with which they do business is constantly changing.

The existing technical solution

The manufacturer currently has a custom-built supplier purchasing system. The system provides a trading hub into which suppliers can integrate, and a supplier portal that enables suppliers to continue with their (largely manual) existing processes. The interactions fall into three tiers:

- Tier one: XML message exchanges.
- Tier two: FTP and similar tools.
- Tier three: Fax.

Tiers one and two are conducted over a combination of leased lines and Virtual Private Networks (VPNs).

The business problem

The custom purchasing system has had disappointing take-up by the suppliers, and as a result the manufacturer continues to face large costs associated with the manual processes that it still uses with the vast majority of its suppliers. A major barrier to adoption of the new system by suppliers has been the cost of integrating supplier systems with the trading hub.

The current solution also has costs for the manufacturers. In particular, the use of leased lines is expensive, and the use of VPNs is difficult to manage when scaled to a large number of suppliers.

The solution when using WS-ReliableMessaging

By using WSDL to describe the services, and by using web services standards to implement the services, you reduce the costs incurred by the manufacturer and suppliers:

- The prevalence of support for web services amongst diverse software vendors makes it easier and cheaper for suppliers to exploit this technology.
- The familiarity of the developer community with web services technologies (and WSDL in particular), and the rich tool support, means that the use of WSDL-described messages make the message schemas easier to use in the context of web services.

- WS-ReliableMessaging is used to ensure that messages are reliably delivered, and duplicate messages are eliminated.
- Web Services Security technologies enable secure interactions across the Internet, without requiring leased lines and VPNs.

Interoperation with other WS-ReliableMessaging providers: use pattern

Web services enable interoperability between heterogeneous platforms. This requirement arises whenever an organization finds itself with applications on one platform (for example WebSphere Application Server) that must work with applications on another platform, whether as a result of merger and acquisition activity, of a deliberate multi-vendor strategy, or as a result of independent software purchasing decisions taken in different parts of the business.

The existing technical solution

A variety of technical solutions exist for application-to-application integration between WebSphere Application Server and other environments. Most of these involve the use of additional third-party or IBM software to facilitate the integration.

More recently, the introduction of web services support has made interoperability possible without the use of additional components.

The business problem

Basic web services support (using SOAP over HTTP) does enable interoperability, but has the following limitations:

- Reliability: The absence of a WS-ReliableMessaging implementation means that the application logic needs to be extended to handle lost or duplicated messages.
- Flexibility: The absence of asynchronous support for web services means that support is limited to synchronous interactions.

Note: Although both request and reply and one-way messaging are supported in earlier version of WebSphere Application Server, they were implemented in a synchronous fashion. This meant that when a web services client invoked a service it did not receive control back from the middleware until after the service application endpoint had been invoked.

The absence of asynchronous, reliable support for web services often leads you to use one of the other approaches, involving additional components. The additional components often use proprietary communication channels or APIs.

The solution by using WS-ReliableMessaging

The addition of WS-ReliableMessaging support to WebSphere Application Server and to other environments enables you to develop reliable asynchronous web services on both platforms. These services should interoperate without additional IBM or third-party components or proprietary bindings.

WS-ReliableMessaging sequences

The WS-ReliableMessaging protocol relies on a “sequence” to manage the transmission of messages from reliable messaging source to reliable messaging destination. Each application message is given an identifier that identifies both the sequence and the message number (the position) within the sequence. Protocol flows are used to create sequences, to acknowledge messages and to terminate sequences.

You can think of a sequence as being a structured conversation between the reliable source and the reliable destination, through which each message in the sequence is passed reliably. A sequence also passes on the set of messages in the sequence in the order in which it receives them, so if it is important that messages are processed in a particular order - for example, if money must be credited to a bank account before a debit instruction is received to pay for a purchase - then those messages should be included in the same sequence.

The developer of a reliable web services application does not have to be aware of sequences, but the system administrator needs to monitor and manage sequences, as described in Detecting and fixing problems with WS-ReliableMessaging.

WS-ReliableMessaging - terminology

Foreign destination
A foreign destination is a software agent, outside of the system, that receives messages reliably through WS-ReliableMessaging.

Foreign source
A Foreign source is a software agent, outside of the system, that sends messages reliably through WS-ReliableMessaging.

Implementation WSDL
A WSDL document that describes not only the interface to a service (that is the messages, port types and bindings) but also the service implementation (that is, it has service and port elements). The service interface should preferably be defined by importing a separate *interface WSDL* document.

Interface WSDL
A WSDL document that describes only the interface to a service. That is, it defines messages, port types and bindings, but not services and ports.

Policy set instance document
A document containing configuration details for a selection of web services standards. For more information, see Securing web services applications with policy sets by using the administrative console

Provider application
A provider application is an application that implements a service.

Reliable web service provider
See also the general definition of a *provider application*. The Reliable web service provider is a provider application that has been deployed into the system being modeled, and is configured to use WS-ReliableMessaging. This is the WebSphere Application Server-hosted equivalent of a *foreign destination*. It is outside of the boundary of the system and interacts with the system.

Reliable web service requester
See also the general definition of a *requester application*. The Reliable web service requester is a requester application that has been deployed into the system being modeled, and is configured to use WS-ReliableMessaging. This is the WebSphere Application Server-hosted equivalent of a *foreign source*. It is outside of the boundary of the system and interacts with the system.

Requester application
A requester application is an application that makes web service requests.

Terminate (a sequence)
When the reliable messaging source has completed its use of a sequence, it sends a TerminateSequence message to the reliable messaging destination to indicate that the sequence is complete and that it will not be sending any further messages related to the sequence. The reliable messaging destination can then safely reclaim any resources associated with the sequence.

WS-ReliableMessaging: supported specifications and standards

WebSphere Application Server provides support for two levels of the WS-ReliableMessaging specification. This gives compatibility with vendors that provide WS-ReliableMessaging support at the February 2005 level, as well as meeting the requirements of the current OASIS specification. This implementation of WS-ReliableMessaging also composes with many other web services standards.

Details of the supported WS-ReliableMessaging specifications are available at the following web addresses:

- The WS-ReliableMessaging specification Version 1.0, February 2005.
- The OASIS WS-ReliableMessaging specification Version 1.1, February 2007.

Support for the WS-ReliableMessaging standard was first introduced as part of the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services. At that time, the Reliable Asynchronous Messaging Profile (RAMP) Version 1.0 specification used WS-ReliableMessaging to ensure the reliable delivery of messages, and the Feature Pack for Web Services in WebSphere Application Server Version 6.1 included default policy sets that support this specification. You can migrate WebSphere Application Server Version 6.1 WS-ReliableMessaging configurations that use RAMP-based policy sets to the current version of the product.

Following on from the RAMP Version 1.0 specification, the Web Services Interoperability organization (WS-I) Reliable Secure Profile working group has developed Version 1.0 of an interoperability profile dealing with secure, reliable messaging capabilities for web services. This profile is similar to RAMP Version 1.0, except that it is updated to use WS-ReliableMessaging Version 1.1 with the OASIS WS-SecureConversation Version 1.3 specification. The WS-I RSP default policy sets provided in this version of WebSphere Application Server are an implementation of the Reliable Secure Profile Version 1.0 specification.

The extent to which WS-ReliableMessaging composes with other web services standards is described in the following sections:

- "WS-Addressing"
- "WS-AtomicTransactions"
- "WS-MakeConnection" on page 845
- "WS-Notification" on page 845
- "WS-Policy" on page 845
- "WS-SecureConversation" on page 846
- "WS-Security" on page 846

WS-Addressing

The WS-ReliableMessaging specification uses WS-Addressing, and the implementation fully supports the asynchronous request and reply model given in the WS-Addressing specification.

Note: WS-ReliableMessaging Version 1.1 messaging requires WS-Addressing to be mandatory. If you use a policy set that includes WS-ReliableMessaging and WS-Addressing policies, and the WS-Addressing policy is configured as optional, then WebSphere Application Server overrides the WS-Addressing setting and automatically enables WS-Addressing.

WS-AtomicTransactions

WS-ReliableMessaging transactions do not use the WS-AtomicTransactions protocol. The relationship between these two protocols is as follows:

- WS-AtomicTransactions and WS-ReliableMessaging are mutually exclusive when WS-ReliableMessaging is being used, with a managed store, to provide transactional recoverable messaging.
- If WS-ReliableMessaging is configured to use an in-memory store, then there are cases where a WS-AtomicTransaction can be flowed between the reliable messaging source and the reliable messaging destination for two-way invocations. In this situation, WS-ReliableMessaging only protects against network failures, not against server failure.

For more information about WS-AtomicTransactions, see Transaction support in WebSphere Application Server. For more information about using WS-ReliableMessaging transactions, see Providing transactional recoverable messaging through WS-ReliableMessaging.

WS-MakeConnection

WS-ReliableMessaging Version 1.1 uses the WS-MakeConnection protocol to enable synchronous message exchange. For more information about this protocol, see the WS-MakeConnection specification Version 1.1, February 28 2008.

WS-MakeConnection uses information contained in WS-Addressing message headers, so for any application that uses reliable synchronous message exchange you must include both WS-ReliableMessaging and WS-Addressing policy in the policy set.

WS-Notification

If you create JAX-WS based WS-Notification services, you can apply WS-ReliableMessaging policies to them to make your WS-Notification services reliable. For more information, see Configuring WS-Notification for reliable notification.

Note: In this release, there are two types of WS-Notification service:

- **Version 7.0**: You configure a Version 7.0 WS-Notification service and service points if you want to compose a JAX-WS WS-Notification service with WS-ReliableMessaging, or if you want to apply JAX-WS handlers to your WS-Notification service. This is the recommended type of service for new deployments.
- **Version 6.1**: You configure a Version 6.1 WS-Notification service and service points if you want to expose a JAX-RPC WS-Notification service by using the same technology provided in WebSphere Application Server Version 6.1, including the ability to apply JAX-RPC handlers to the service.

WS-Policy

The WS-Policy implementation in WebSphere Application Server supports Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1.

You can use the WS-Policy protocol to exchange policies in standard format. You can communicate the policy configuration to any other client, service registry or service that supports the WS-Policy specification, including non-WebSphere Application Server products in a heterogeneous environment. For a service provider, the policy configuration can be shared in published WSDL. For a client, the client can obtain the policy of the service provider in the standard WS-PolicyAttachments format and use this information to establish a configuration that is acceptable to both the client and the service provider. In other words, the client can be configured dynamically, based on the policies supported by its service provider.

At any stage - that is, before or after you have built your reliable web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

WS-SecureConversation

WS-ReliableMessaging is designed to work with WS-SecureConversation. A secure conversation context is established and this is used to secure the application messages and the WS-ReliableMessaging protocol messages.

To use WS-SecureConversation, create or apply a policy set that includes both WS-ReliableMessaging and WS-SecureConversation. For example, either of the WS-I RSP default policy sets.

WS-Security

WS-ReliableMessaging composes with WS-Security. The WS-ReliableMessaging headers appended to application messages are signed if required. The WS-ReliableMessaging protocol messages are signed and encrypted if required.

Security processing is done close to the transport: after WS-ReliableMessaging processing at the web service requester and before WS-ReliableMessaging processing at the web service provider. This means the messages held in the WS-ReliableMessaging store are not signed and encrypted, so the emphasis is on the administrator to secure the store, if the store being used is the messaging engine in a service integration bus.

Note: If possible, use WS-SecureConversation rather than WS-Security because the WS-SecureConversation protocol is less susceptible to security attacks.

WS-ReliableMessaging roles and goals

Computing roles that members of your organization might perform, and how you can use WS-ReliableMessaging to help meet the goals of each role.

For a general description of each of the following roles, see Chapter 36, “WebSphere Application Server roles and goals,” on page 1071.

Application developer

The application developer is responsible for creating the WS-ReliableMessaging requester applications and provider applications. The application developer is responsible for writing the application code and packaging the application into a deployable unit.

Developing

- Goal: Develop a JAX-WS Web service application.

System administrator

The system administrator is responsible for providing the infrastructure, such as configured application servers, through which the applications can be deployed and executed. The system administrator is also responsible for deploying applications into the configured environment, and for maintaining the environment and applications in good working order. This maintenance role includes operational management of state, such as messages and transaction state, and problem diagnosis to determine message or transaction outcomes.

Deploying

- Goal: Configure a policy set instance to enable WS-ReliableMessaging.
- Goal: Install your reliable JAX-WS web service application.

- Goal: Attach and bind a WS-ReliableMessaging policy set to your application.

Operating

- Goal: Detect and fix problems with WS-ReliableMessaging.

WS-ReliableMessaging - requirements for interaction with other implementations

The information and configuration that is needed for another vendor's reliable messaging source to send messages to a WebSphere Application Server reliable messaging destination, or for a WebSphere Application Server reliable messaging source to send messages to another vendor's reliable messaging destination.

Using another vendor's reliable messaging source to send messages to a WebSphere Application Server reliable messaging destination

For another vendor's WS-ReliableMessaging implementation to interact with a WebSphere Application Server WS-ReliableMessaging endpoint, the other vendor's reliable messaging source needs to know the target endpoint address and port for the WebSphere Application Server application that is enabled for reliable messaging. The WS-ReliableMessaging protocol messages are sent to the same endpoint address as the application messages. You can get this address from the WSDL published by the WebSphere Application Server endpoint.

The reliable messaging source controls the endpoint reference used for acknowledgement messages, so the other vendor's product might have to use the WS-Addressing anonymous URI. For more information, see WS-ReliableMessaging - How it works. Whether or not the reliable messaging source uses the WS-Addressing anonymous URI, the WebSphere Application Server reliable messaging destination can work with the reliable messaging source without further configuration.

A WebSphere Application Server reliable messaging destination cannot tell whether the reliable messaging source is durable and transactional. If you want durable transactional web services, check that the other vendor's reliable messaging source supports that mode of operation, as well as configuring the WebSphere Application Server end of the link.

Using a WebSphere Application Server reliable messaging source to send messages to another vendor's reliable messaging destination

For an application in WebSphere Application Server to invoke a web service that uses WS-ReliableMessaging, the information required is the target endpoint address and port for the web service being invoked. The WS-ReliableMessaging protocol messages are sent to the same endpoint address as the application messages. You can usually get this address from the WSDL published by the target Web service.

The WebSphere Application Server Source is provided with additional WS-ReliableMessaging configuration, modeled as part of the policy set associated with the web service client. The policy set might configure the reliable messaging source to use the WS-Addressing anonymous URI as the address within the endpoint reference used for acknowledgement messages. For more information, see “WS-ReliableMessaging - How it works” on page 836

WebSphere Application Server cannot tell whether the reliable messaging destination is durable and transactional. If you want durable transactional web services, check that the other vendor's reliable messaging destination supports that mode of operation, as well as configuring the WebSphere Application Server end of the link.

WS-Transaction

WS-Transaction is an interoperability standard that includes the WS-AtomicTransaction, WS-BusinessActivity, and WS-Coordination specifications. Use these topics to learn more about WS-Transaction.

The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the web services environment. Distributed web services applications, and the resources they use, can take part in distributed global transactions. With Web Services Business Activity (WS-BA) support in the application server, web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs. Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant web services can enlist to take part in the protocols that are offered by specific coordination types.

To learn about WS-Transaction in WebSphere Application Server, see the following topics:
- "Web Services Atomic Transaction support in the application server"
- "Web Services Business Activity support in the application server" on page 852
- "Web services transactions, high availability, firewalls and intermediary nodes" on page 854
- "Transaction compensation and business activity support" on page 122
- "WS-Transaction and mixed-version cells" on page 860

Web Services Atomic Transaction support in the application server

The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the web services environment. Distributed web services applications, and the resources they use, can take part in distributed global transactions.

Web services protocols provide standard ways of defining web services applications, allowing the applications to operate independently of the product, platform, or programming language that is used. The WS-AT support is an implementation of the following specifications on the application server. These specifications define a set of web services that enable web services applications to participate in global transactions that are distributed across the heterogeneous web services environment.
- WS-AT is a specific coordination type that defines protocols for atomic transactions. The specifications are:
  - Web Services Atomic Transaction Version 1.0
  - Web Services Atomic Transaction Version 1.1
  - Web Services Atomic Transaction Version 1.2
- Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant web services can enlist to take part in the protocols that are offered by specific coordination types. The specifications are:
  - Web Services Coordination Version 1.0
  - Web Services Coordination Version 1.1
  - Web Services Coordination Version 1.2

The WS-AT support is support for an interoperability protocol that introduces no new programming interfaces for transactional support. Global transaction demarcation is provided by standard enterprise application use of the Java Transaction API (JTA) UserTransaction interface. If an application component that is running under a global transaction makes a web services request, a WS-AT CoordinationContext is implicitly propagated to the target web service, but only if the appropriate application deployment descriptors have been set, as described in the topic about configuring transactional deployment attributes.

If the application server is the system hosting the target endpoint for a web services request that contains a WS-AT CoordinationContext, the application server automatically establishes a subordinate JTA transaction in the target runtime environment that becomes the transactional context under which the target web service application runs.

The following figure, shows a transaction context shared between two application servers for a web services request that contains a WS-AT CoordinationContext.

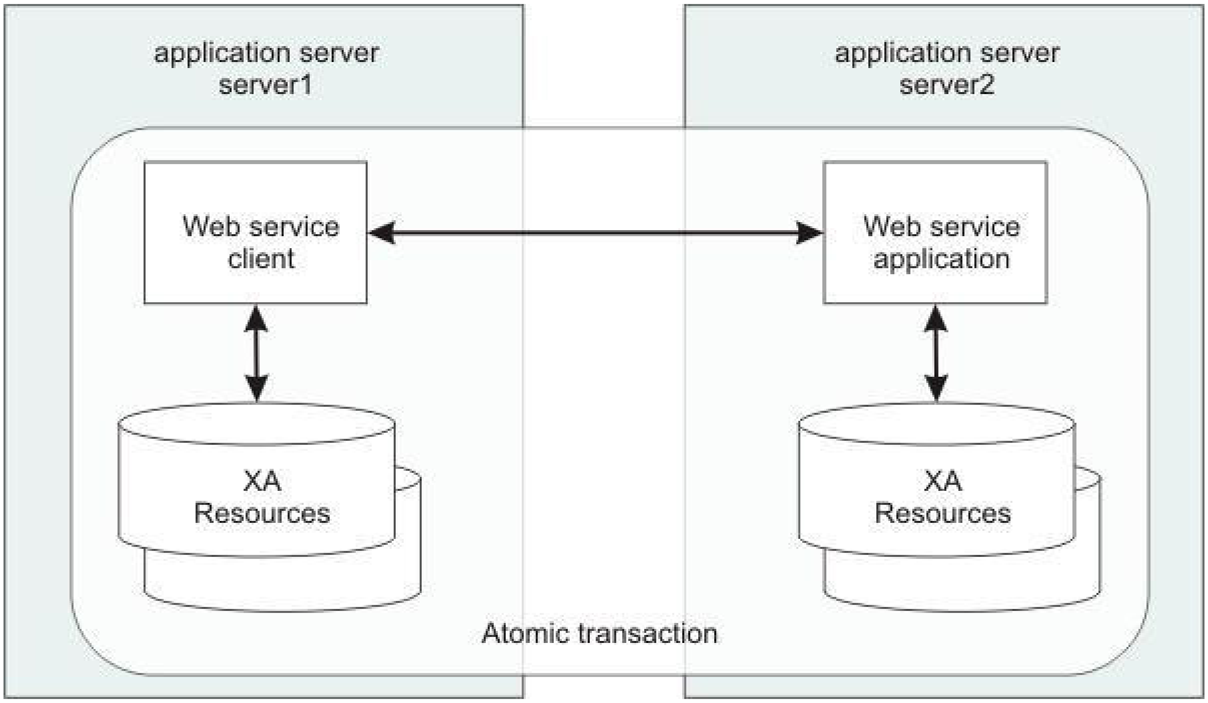


Figure 116. Transaction context shared between two application servers

You can configure the policies for the WS-AtomicTransaction protocol. You can configure whether a client propagates, and a server receives, a WS-AT context. To ensure that a client always sends WS-AtomicTransaction context when it makes an outbound service request, you must associate a policy set with the client, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Mandatory. Alternatively, if you know that the client always invokes remote endpoints that include the WS-AtomicTransaction ATAssertion policy type attribute, you can configure the client to apply the WS-Policy configuration of the provider so that the client adopts the mandatory policy of the provider automatically.

To ensure that any requests that a web services provider receives include a WS-AtomicTransaction context, you must associate a policy set with the provider, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Mandatory.

To ensure that a client or provider never use WS-AtomicTransaction context, you must associate a policy set with the client or provider, where the policy set includes the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Never. You might use this configuration for environments where you do not want web services requests to create a tight coupling between a client and a provider, for example when there are requests between enterprises.

If there is no policy set associated with a client or provider, or the WS-Transaction policy type is not included in the policy set, the default WS-Transaction behavior is used.

WS-AT support restrictions

In this version of the application server, WS-AT contexts cannot be started from a non-recoverable client process.

Application design

WS-AT is a two-phase commit transaction protocol and is suitable for short duration transactions only.

An atomic transaction coordinates resource managers that isolate transactional updates by holding transactional locks on resources. Therefore, it is generally not recommended that WS-AT transactions are distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit, and in such scenarios, it can be more appropriate to use a compensating business transaction, for example, as part of a Business Process Execution Language (BPEL) process, or by using Web Services Business Activity (WS-BA).

WS-AT is most appropriate for distributing transaction context across web services that are deployed in a single enterprise. Only request-response message exchange patterns carry transaction context because the originator (application or container) of a transaction must be sure that all business tasks that are run under that transaction have finished before requesting the completion of a transaction. Web services invoked by a one-way request never run under the transaction of the requesting client.

The effect of service faults on WS-AT transactions is similar to the effect of Enterprise JavaBeans (EJB) application exceptions on transactions, as described in the EJB specification. If a service that is running under a requester WS-AT transaction returns a fault, the application server does not automatically mark the transaction rollback-only. The exception handler of the requester chooses whether the transaction can progress and chooses whether to mark the transaction rollback-only. If the requester is running in the application server, the standard JTA or EJB APIs can be used to mark the transaction rollback-only. The service component that generates the fault might, itself, mark the transaction rollback-only before returning the fault. If the implementation of the service component encounters a system exception, it typically allows its container to handle the exception. Application server containers automatically mark any received transaction context rollback-only when handling a system exception that is generated by a service implementation.

Application development

There are no specific development tasks required for web services applications to take advantage of WS-AT.

For JAX-RPC applications, there are some application deployment descriptors that you must set appropriately, as described in the topic about configuring transactional deployment attributes. The JAX-RPC run time supports WS-AT 1.0.

For JAX-WS applications, enable WS-AT support by creating a policy set, adding the WS-Transaction policy type to the policy set, optionally configuring the policy type, and attaching the policy set to the application or client that will be involved in the WS-AT communication. The JAX-WS run time supports WS-AT 1.0, WS-AT 1.1, WS-AT 1.2 and the WS-Policy assertion for WS-AT.

When the JAX-WS runtime receives an inbound request, WS-Transaction 1.0, WS-Transaction 1.1 and WS-Transaction 1.2 specification levels are supported. When an outbound JAX-WS request is sent, only one specification level can be used. If WS-Transaction WS-Policy assertions are available, either from the Web Services Description Language (WSDL) of the target web service, or from the WS-Transaction policy type of the client, the specification level that is applicable to the client and the target web service is used. For example, if the hosting environment of the target web service supports only WS-Transaction 1.0, WS-AT 1.0 is used. If both specification levels are applicable, or if no WS-Transaction WS-Policy assertions are available, the default WS-Transaction specification level that is set in the Transaction service settings is used.

The default behavior when there is no effective policy set, or when the WS-Transaction policy type is not included in the effective policy set, is as follows:

- For a client, if the client does not consider the policy of the provider, the client does not send any WS-AT or Web Services Business Activity (WS-BA) context. This behavior is equivalent to a WS-Transaction policy setting of Never.
- For a client, if the client considers the policy of the provider, the client sends WS-AT or WS-BA context if the policy of the provider includes WS-AT or WS-BA assertions. This behavior is equivalent to a WS-Transaction policy setting of Supports.
- For a server, the server does not receive any WS-AT or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.

Application developers do not have to explicitly register WS-AT participants. The application server run time takes responsibility for the registration of WS-AT participants, in the same way as the registration of XAResources in the JTA transaction to which the WS-AT transaction is federated. At transaction completion time, all XAResources and WS-AT participants are coordinated atomically by the application server transaction service.

If a JTA transaction is active on the thread when a web services application request is made, the transaction is propagated across on the web services request and established in the target environment. This process is similar to the distribution of transaction context over IIOP, as described in the EJB specification. Any transactional work performed in the target environment becomes part of the same global transaction.

WS-Transaction policy assertions

If you configure the policies for the WS-Transaction protocol for a provider, this configuration affects the assertions that are included in any WSDL that is generated for the web service with which the policy type is associated. The WS-Policy assertion that is used to describe the transactional requirements of a client or provider that uses WS-AtomicTransaction is ATAssertion. If the WS-Transaction policy type has a WS-AtomicTransaction setting of Mandatory or Supports, a policy assertion is included in the WSDL.

The application server can also parse, understand, and apply such assertions that are in WSDL that it parses.

The following example shows WSDL where the WS-AtomicTransaction ATAssertion indicates that an endpoint must be invoked with WS-AT context included in the request message, and that the context can be in WS-Transaction 1.0 or 1.1 format. There are two namespaces and there are two assertions; one for each WS-Transaction specification level, that use the WS-Policy ExactlyOne operator to show that the client must choose which specification level to use.

```
<wsdl:definitions targetNamespace="bank.example.com"
     xmlns:tns="bank.example.com"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wsat11="http://docs.oasis-open.org/ws-tx/wsat/2006/06"
     xmlns:wsat10="http://schemas.xmlsoap.org/ws/2004/10/wsat"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <wsp:Policy wsu:Id="ATPolicy">
         <wsp:ExactlyOne>
             <wsat11:ATAssertion />
             <wsat10:ATAssertion />
             <!-- omitted assertions -->
         </wsp:ExactlyOne />
     </wsp:Policy>
     <!-- omitted elements -->
     <wsdl:binding name="BankBinding" type="tns:BankPortType">
```

```
        <!-- omitted elements -->
        <wsdl:operation name="TransferFunds">
            <wsp:PolicyReference URI="#ATPolicy" wsdl:required="true"/>
            <!-- omitted elements -->
        </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

Web Services Business Activity support in the application server

With Web Services Business Activity (WS-BA) support in the application server, web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs.

Web services protocols are defined by the Organization for the Advancement of Structured Information Standards (OASIS) group and provide standard ways of defining web services applications, allowing the applications to operate independently of the product, platform or programming language that is used. The Web Services Business Activity (WS-BA) support is an implementation of the following specifications in the application server. These specifications define a set of protocols that enable web services applications to participate in loosely coupled business processes that are distributed across the heterogeneous web services environment, with the ability to compensate actions if an error occurs. For example, an application that sends an email cannot retrieve that email following a failure in the business task. However, the application can provide a business-level compensation handler that sends another email advising of the new circumstances. A *business activity* is a group of general tasks that you want to link together so that the tasks have an agreed outcome.

- WS-BA is a specific coordination type that defines protocols for business activities. The specifications are:
  - Web Services Business Activity Version 1.0
  - Web Services Business Activity Version 1.1
  - Web Services Business Activity Version 1.2
- Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant web services can enlist to take part in the protocols that are offered by specific coordination types. The specifications are:
  - Web Services Coordination Version 1.0
  - Web Services Coordination Version 1.1
  - Web Services Coordination Version 1.2

In addition to supporting the WS-BA interoperability protocol, the application server provides a programming interface for creating business activities and compensation handlers. With this programming interface, you can specify compensation data and check or alter the status of a business activity.

You can also use this compensation facility with applications that are not web services, as long as these applications involve communication between WebSphere Application Servers only. See the related topics for more information.

You can configure the policies for the WS-BusinessActivity protocol. You can configure whether a client propagates, and a server receives, a WS-BA context. To ensure that a client always sends WS-BusinessActivity context when it makes an outbound service request, you must associate a policy set with the client, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Mandatory. Alternatively, if you know that the client always invokes remote endpoints that include the WS-BusinessActivity BAAtomicOutcomeAssertion policy type attribute, you can configure the client to apply the WS-Policy configuration of the provider so that the client adopts the mandatory policy of the provider automatically.

To ensure that any requests that a web services provider receives includes a WS-BusinessActivity context, you must associate a policy set with the provider, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Mandatory.

To ensure that a client or provider never use WS-BusinessActivity context, you must associate a policy set with the client or provider, where the policy set includes the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Never. You might use this configuration for environments where you do not want web services requests to create a tight coupling between a client and a provider, for example when there are requests between enterprises.

If no policy set is associated with a client or provider, or the WS-Transaction policy type is not included in the policy set, the default WS-Transaction behavior is used.

Application development

No specific development tasks are required for web services applications to take advantage of WS-BA.

For JAX-RPC applications, any Enterprise JavaBeans (EJB) component that is configured to run under a BusinessActivity scope automatically propagates that scope when it makes an outbound JAX-RPC web services request. The JAX-RPC run time supports WS-BA 1.0.

For JAX-WS applications, enable WS-BA support by creating a policy set, adding the WS-Transaction policy type to the policy set, optionally configuring the policy type, and attaching the policy set to the application or client that will be involved in the WS-BA communication. The JAX-WS run time supports WS-BA 1.0, WS-BA 1.1, WS-BA 1.2, and the WS-Policy assertion for WS-BA.

When the JAX-WS run time receives an inbound request, WS-Transaction 1.0, WS-Transaction 1.1 and WS-Transaction 1.2 specification levels are supported. When an outbound JAX-WS request is sent, only one specification level can be used. If WS-Transaction WS-Policy assertions are available, either from the Web Services Description Language (WSDL) of the target web service, or from the WS-Transaction policy type of the client, the specification level that is applicable to the client and the target web service is used. For example, if the hosting environment of the target web service supports only WS-Transaction 1.0, WS-BA 1.0 is used. If both specification levels are applicable, or if no WS-Transaction WS-Policy assertions are available, the default WS-Transaction specification level that is set in the Transaction service settings is used.

The default behavior when there is no effective policy set, or when the WS-Transaction policy type is not included in the effective policy set, is as follows:

- If a client does not consider the policy of the provider, the client does not send any Web Service Atomic Transaction (WS-AT) or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.
- If a client does consider the policy of the provider, the client sends WS-AT or WS-BA context if the policy of the provider includes WS-AT or WS-BA assertions. This behavior is equivalent to a WS-Transaction policy configuration setting of Supports.
- A server does not receive any WS-AT or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.

WS-Transaction policy assertions

If you configure the policies for the WS-BusinessActivity protocol for a provider, this affects the assertions that are includes in any WSDL that is generated for the web service with which the policy type is associated. The WS-Policy assertion that is used to describe the compensation requirements of a client or provider that uses WS-BusinessActivity is BAAtomicOutcomeAssertion. If the WS-Transaction policy type has a WS-BusinessActivity setting of Mandatory or Supports, a policy assertion is included in the WSDL.

The application server can also parse, understand, and apply such assertions that are in WSDL that it parses.

The following example shows WSDL where the WS-BusinessActivity BAAtomicOutcomeAssertion indicates that an endpoint must be invoked with WS-BA context included in the request message, and that the context can be in WS-Transaction 1.0 or 1.1 format. There are two namespaces and two assertions; one for each WS-Transaction specification level, that use the WS-Policy ExactlyOne operator to show that the client must choose which specification level to use.

```
<wsdl:definitions targetNamespace="bank.example.com"
     xmlns:tns="bank.example.com"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wsat11="http://docs.oasis-open.org/ws-tx/wsba/2006/06"
     xmlns:wsat10="http://schemas.xmlsoap.org/ws/2004/10/wsba"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <wsp:Policy wsu:Id="BAPolicy">
         <wsp:ExactlyOne>
             <wsat11:BAAtomicOutcomeAssertion />
             <wsat10:BAAtomicOutcomeAssertion />
             <!-- omitted assertions -->
         </wsp:ExactlyOne />
     </wsp:Policy>
     <!-- omitted elements -->
     <wsdl:binding name="BankBinding" type="tns:BankPortType">
          <!-- omitted elements -->
          <wsdl:operation name="TransferFunds">
               <wsp:PolicyReference URI="#BAPolicy" wsdl:required="true"/>
               <!-- omitted elements -->
          </wsdl:operation>
     </wsdl:binding>
</wsdl:definitions>
```

Web services transactions, high availability, firewalls and intermediary nodes

You can configure your system to enable propagation of Web Services Atomic Transactions (WS-AT) message contexts and Web Service Business Activities (WS-BA) message contexts across firewalls or outside the WebSphere Application Server domain. With these configurations, you can distribute web service applications that use WS-AT or WS-BA across disparate systems.

The topologies that are available to you are as follows:

Direct connection

No intermediary node exists in this topology. The client communicates directly with the specificWebSphere Application Server on which the target service is hosted.

HTTP server, such as IBM HTTP Server

In this topology, the client communicates with an HTTP server, which always routes the client requests and web services transaction protocol messages to a specific WebSphere Application Server. Also, typically you need to configure the HTTP server for web services transactions, that is, configure it to deliver web services transaction protocol messages to the appropriate WebSphere Application Server.

Transactional integrity is assured, because recovery processing occurs after the failed server restarts.

Transaction compensation and business activity support

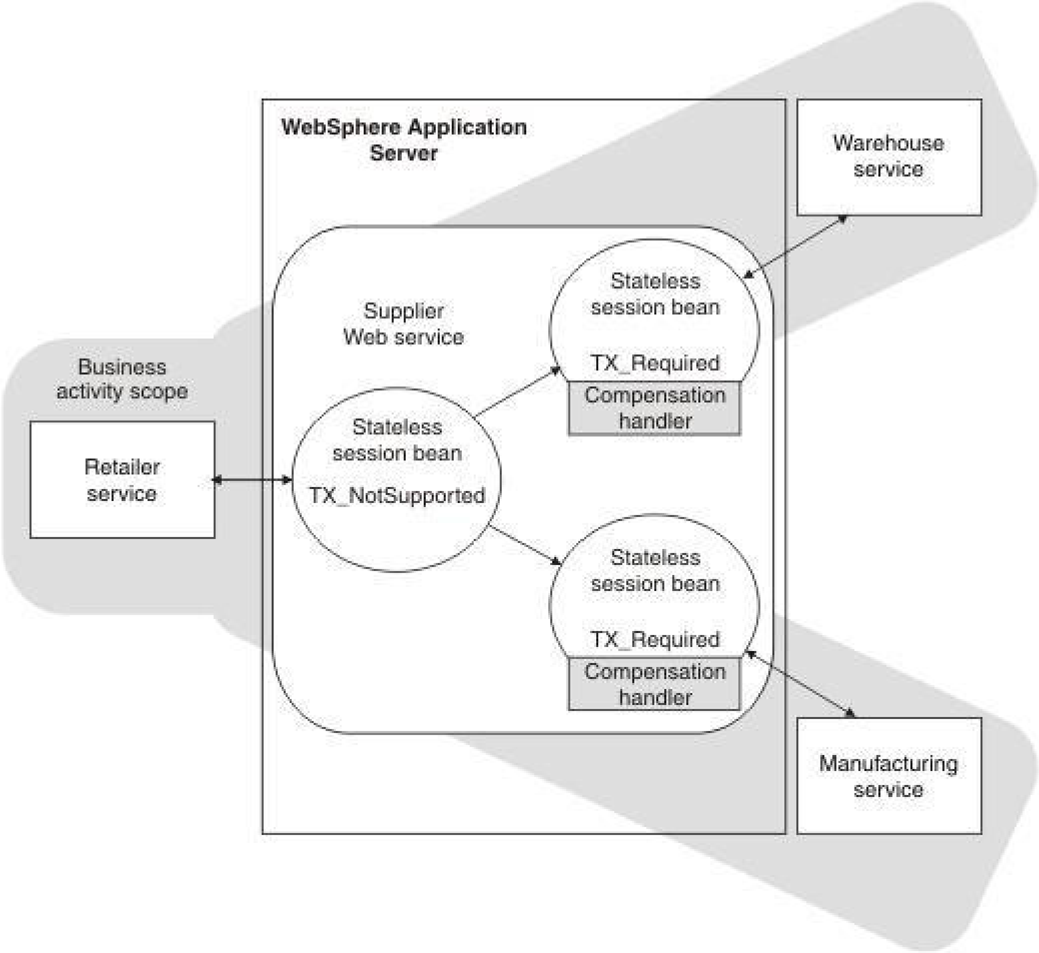
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an email can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Serverenvironments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that undertake work that can be compensated. These other session beans each have a *compensation handler*; a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an email.



Application design

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work by using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functions. Enterprise beans that exploit business activity scopes can offer web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in web service messages by using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

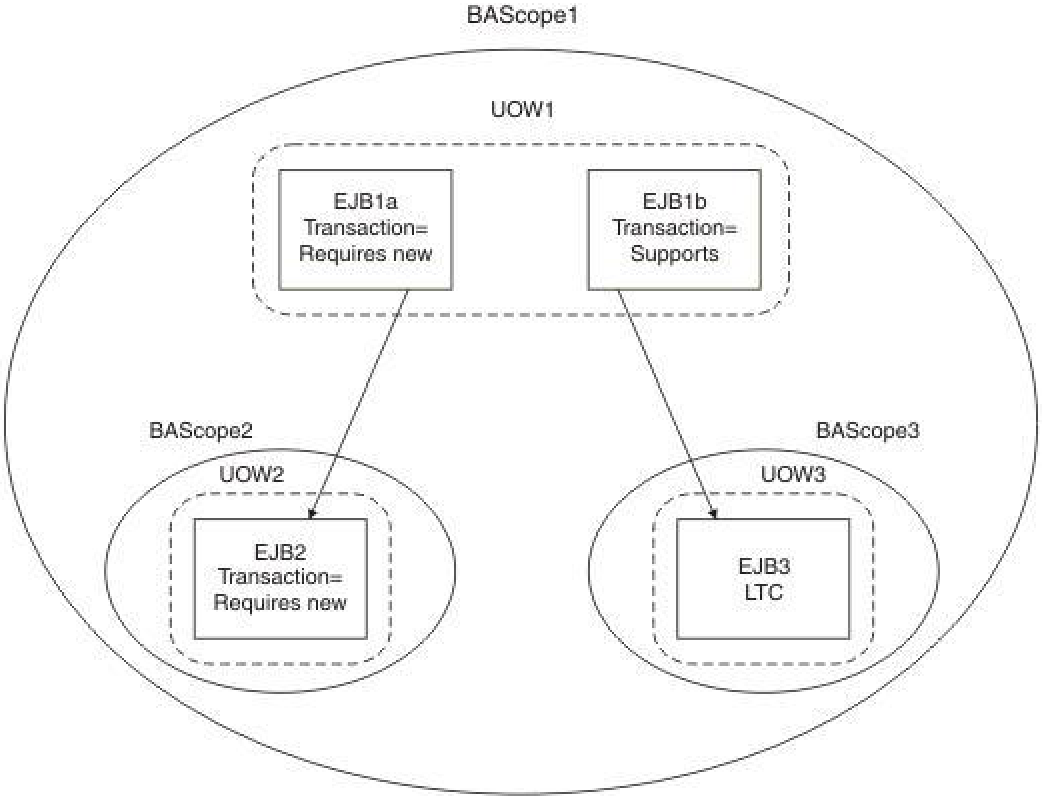
Business activity scopes

The scope of a business activity is that of a main WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing main UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any main UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.



Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by trying the called component again or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 68. Compensation behavior for a single business activity scope. The table lists the possible combinations of success and failure for the inner and outer business activity scopes, and the compensation behavior associated with each combination.

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see the topic about the business activity API.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight by using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration

If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope

Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the

received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method, then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For details, see the topic about creating an application that uses the WS-BA support.

WS-Transaction and mixed-version cells

You must consider WS-Transaction policy type enablement and behavior, and the WS-Transaction specification level to use, when a cell contains servers at different versions; for example, WebSphere Application Server Version 7.0 or later and WebSphere Application Server Version 6.1 Feature Pack for Web Services.

WS-Transaction policy type enablement

For a Version 6.1 Feature Pack for Web Services server, you can enable the WS-Transaction policy type by including it in a policy set, but you cannot configure it. For a Version 7.0 or later server, you can both enable and configure the WS-Transaction policy type. Configuration information is written to the WS-Transaction policy type file.

In a cell with both Version 6.1 Feature Pack for Web Services and Version 7.0 or later servers, the following behavior occurs:

- If a Version 6.1 Feature Pack for Web Services server reads a WS-Transaction policy type file that is generated by a Version 7.0 or later server, the server enables the WS-Transaction policy type, but ignores any configuration information in the file.
- If a Version 7.0 or later server reads a WS-Transaction policy type that is generated by a Version 6.1 Feature Pack for Web Services server, the server enables the WS-Transaction policy type by using a value of Supports for the WS-AtomicTransaction and WS-BusinessActivity protocols. This value is equivalent to the existing behavior of a Version 6.1 Feature Pack for Web Services server.

WS-Transaction specification level

WebSphere Application Server supports the WS-Transaction 1.0, WS-Transaction 1.1 and WS-Transaction 1.2 specifications. In practice, version 1.2 of the WS-Transaction standard is functionally equivalent to version 1.1, so within WebSphere Application Server, wherever WS-Transaction 1.1 is supported, WS-Transaction 1.2 is also.

A WebSphere Application Server Version 6.x server supports WS-Transaction 1.0. A Version 7.0 or later server supports WS-Transaction 1.0, 1.1 and 1.2.

No special restrictions apply to a cell with both Version 6.x and Version 7.0 or later servers.

Business activity API

Use the business activity application programming interface (API) to create business activities and compensation handlers for an application component, and to log data that is required to compensate an activity if there is a failure in the overall business activity.

Overview

The business activity support provides a UserBusinessActivity API and two interfaces: a serializable.CompensationHandler interface and a CompensationHandler interface. Each interface has two exceptions: RetryCompensationHandlerException and CompensationHandlerFailedException. You can look up the UserBusinessActivity interface from the application server Java Naming and Directory Interface (JNDI) at `java:comp/websphere/UserBusinessActivity`. For example:

```
InitialContext ctx = new InitialContext();
UserBusinessActivity uba = (UserBusinessActivity) ctx.lookup("java:comp/websphere/UserBusinessActivity");
```

You can use the **getId** method to access the unique identifier for the business activity that is currently associated with the calling thread. The identifier is the same as the one that is generated for the business activity scope at run time and that is used for information, warning, and error messages. For example, the application can use the identifier in audit or diagnostic messages, and it is possible to correlate between application-generated and runtime-generated messages.

```
InitialContext initialContext = new InitialContext();
UserBusinessActivity uba = initialContext.lookup("java:comp/websphere/UserBusinessActivity");
...
String activityId = uba.getId();
if (activityId == null)
// No activity on the thread
else
// Output audit message including activity id
```

If an application component runs work that might require compensating upon failure in the business activity, you must provide a compensation handler class that is assembled as part of the deployed application. This Java class must implement one of the following interfaces:

- com.ibm.websphere.wsba.serializable.CompensationHandler, which takes a takes a parameter of a serializable object
- com.ibm.websphere.wsba.CompensationHandler, which takes a parameter of a Service Data Object (SDO)

Typically, applications that already have their data available in DataObject format will use the CompensationHandler interface, and applications that do not will use the serializable.CompensationHandler interface. Both interfaces support the close and compensate methods.

An application must register a compensation handler implementation that works with the type of compensation data (serializable object or SDO) that the application uses. If there is a mismatch between the type of data that the application component uses and the compensation handler implementation, there is an error.

During normal application processing, the application can make one or more invocations to the `setCompensationDataImmediate` or `setCompensationDataAtCommit` methods, passing in either a serializable object or an SDO that represents the current state of the work performed.

When the underlying unit of work (UOW) that the root business activity is associated with completes, all registered compensators are coordinated to complete. During completion, either the compensate or the close method is called on the compensation handler, passing in the most recent compensation data logged by the application component as a parameter. Your compensation handler implementation must be able to understand the data that is stored in either the serializable object or the SDO DataObject; when using this data, the compensation handler must be able to determine the nature of the work performed by the enterprise bean and compensate or close in an appropriate way, for example by undoing changes made to database rows if there is a failure in the business activity. You associate the compensation handler with an application component by using the assembly tooling, such as Rational Application Developer.

Active and inactive compensation handlers

You implement the serializable.CompensationHandler or CompensationHandler interface for any application component that executes code that might have to be compensated within a business activity scope. Compensation handler objects are registered implicitly with the business activity scope under which the application runs, whenever the application calls the UserBusinessActivity API to specify compensation data. Compensation handlers can be in one of two states, active or inactive, depending on any transactional UOW under which they are registered. A compensation handler that is registered within a transactional UOW might initially be inactive until the transaction commits, at which point the compensation handler becomes active (see the following section). A compensation handler that is registered outside a transactional UOW always becomes active immediately.

When a business activity completes, it drives only active compensation handlers. Any inactive compensation handlers that are associated with the business activity are discarded and never driven.

Logging compensation data

The business activity API specifies two methods that allow the application to log compensation data. This data is made available to the compensation handlers during their processing when the business activity completes. The application calls one of these methods, depending on whether it expects transactions to be part of the business activity.

setCompensationDataAtCommit()

Call the setCompensationDataAtCommit method when the application expects a global transaction on the thread.

- If a global transaction is present on the thread, the CompensationHandler object is initially inactive. If the global transaction fails, it rolls back any transactional work done within its transaction context in an atomic manner, and drives the business activity to compensate other completed UOWs. The compensation handler does not have to be involved. If the global transaction commits successfully, the compensation handler becomes active because if the overall business activity fails, the compensation handler is required to compensate the durable work that is completed by the global transaction. The setCompensationDataAtCommit method configures the CompensationHandler instance to undertake this compensation function.
- If a global transaction is not present when the setCompensationDataAtCommit method is called, the compensation handler becomes active immediately.

For example, for an SDO, and using the same business activity instance as in the previous example:

```
DataObject compensationData = doWorkWhichWouldNeedCompensating();
uba.setCompensationDataAtCommit(compensationData);
```

setCompensationDataImmediate()

Call the setCompensationDataImmediate method when the application does not expect a global transaction on the thread.

The setCompensationDataImmediate method makes a CompensationHandler instance active immediately, regardless of the current UOW context at the time that the method is invoked. The compensation handler is always able to participate during completion of the business activity.

The role of the setCompensationDataImmediate method is to compensate any non-transactional work, in other words, work that can be performed either inside or outside a global transaction, but that is not governed by the transaction. An example of this type of work is sending an email. The compensation handler must be active immediately so that if a failure occurs in a business activity, this non-transactional work is always compensated.

For example, for a serializable object, and using the same business activity instance as in the previous example:

```
Serializable compensationData = new MyCompensationData();
uba.setCompensationDataImmediate(compensationData);
```

Although these two compensation data logging methods, if called in the same enterprise bean, use the same compensation handler class, they create two separate instances of the compensation handler class at run time. Therefore, the actions of the methods are mutually exclusive; calling one of the methods does not overwrite any work carried out by the other method.

If a compensation handler instance is already added to the Business Activity by using one of these methods, and then the same method is called, passing in `null` as a parameter, that compensation handler instance is removed from the business activity, and is not driven to close or compensate during completion of the business activity.

As described previously, the business activity support adds a compensation handler instance to the business activity when a compensation data logging method is called for the first time by the enterprise bean that uses that business activity. At the same time, a snapshot of the enterprise application context is taken and logged with the compensation data. When the business activity competes, all the compensation handlers that were added to the business activity are driven to compensate or close. The code that you create in the CompensationHandler or serializable.CompensationHandler class is guaranteed to run in the same enterprise application context that was captured in the earlier snapshot.

For details about the methods available in the business activity API, see the topic about additional APIs.

Overview of the Version 3 UDDI registry

The Universal Description, Discovery, and Integration (UDDI) specification defines a way to publish and discover information about web services.

You can find the UDDI specification on the OASIS UDDI web page.

The UDDI specification defines a standard for the visibility, reusability, and manageability that are essential for a service-oriented architecture (SOA) registry service.

The UDDI registry is a directory for web services that is implemented using the UDDI specification. It is a component of WebSphere Application Server.

The UDDI registry is a critical component of the IBM on-demand service-oriented architecture. It solves the problem of discovery of technical components for an enterprise and its partners in the following ways:

- The UDDI registry provides control, flexibility, and confidentiality so that an enterprise can protect its e-business investments.
- The UDDI registry increases efficiency by making it easier to identify technical assets.
- The UDDI registry leverages existing infrastructures

The following example shows how the UDDI registry can be used in a larger enterprise.

A company has an existing application that provides telephone numbers and human resources (HR) information about employees. This application is turned into a web service and published to the registry. A developer in the same company wants to write an application for a procurement function that also needs to provide HR information to the supplier. The application needs to give the supplier access to the employee account codes after the employee provides a name or serial number. Before web services, the developer might be in one of the following situations:

- The developer does not know about the similar application.
- The developer knows about the application, but cannot reuse it because of technical barriers.
- The developer knows about the application and reuses it, but only after significant time and negotiation.

With UDDI, the developer can search for the web service and reuse the existing technical component in their new application for the supplier in minutes. The developer saves time and gets the application running sooner, thereby increasing efficiency and saving the company time and money. The UDDI registry was the first version 2 standard-compliant UDDI registry for private enterprise work. The UDDI registry in this version has the following characteristics:

- It supports the UDDI Version 3.0 specification, in addition to the Version 1.0 and Version 2.0 standard APIs.

- It leverages the proven, reliable WebSphere Application Server technology.
- It uses a relational database, such as DB2, for its persistent store.

What is new in UDDI Version 3

The main aspects of the UDDI Version 3 specification that are provided with this version of WebSphere Application Server are as follows:

Improved recognition of the importance of private UDDI registries
Private UDDI registries are registries that are installed, owned, managed, and controlled by a separate body such as a department within a company, a company, an industry consortium, or an e-marketplace.

Publisher-assigned keys
The publisher of a UDDI entity can specify its key, rather than the registry automatically assigning a unique key. This means that URI-based keys can be used, and it makes it easier to manage multiple registries.

UDDI information model improvements
The UDDI data structures are extended, which improves the ability of UDDI to represent businesses and services through metadata.

Security enhancements
Digital signatures provide additional security. Each of the main UDDI entities can be digitally signed, which improves the integrity and trustworthiness of UDDI data.

Ownership transfer APIs
These APIs support the transfer of the ownership of a UDDI entity from one publisher to another.

UDDI policy
You can set policy to define the behavior of a UDDI registry and therefore recognize the different environments in which a UDDI registry is used.

HTTP GET support for UDDI entities
You can use HTTP GET to access XML representations of each of the UDDI data structures. This extends the HTTP GET service beyond the scope for discovery URLs in the UDDI Version 2 specification.

Additional UDDI registry capabilities

The Version 3 UDDI registry in this version of WebSphere Application Server provides the following capabilities that are additional to support for the UDDI Version 3 specification:

Version 2 UDDI inquiry and publish SOAP API compatibility
There is compatibility with the Version 1 and Version 2 SOAP Inquiry and Publish APIs.

UDDI administrative console extension
The WebSphere Application Server administrative console includes a section that administrators can use to manage UDDI-specific aspects of their WebSphere environment. This management includes the ability to set defaults for initialization of the UDDI node, such as its node ID, and to set the UDDI Version 3 policy values.

UDDI registry administrative interface
The Java Management Extensions (JMX) administrative interface enables administrators to manage UDDI-specific aspects of the WebSphere environment programmatically.

Multidatabase support
The UDDI data is stored in a registry database. The following database products that are supported by WebSphere Application Server are also supported for use as the persistence store for the UDDI registry. For specific details on supported levels, see Detailed system requirements page.

- Apache Derby
- IBM i DB2 for IBM i
- IBM i Oracle

User-defined value set support

You can create your own categorization schemes or value sets. These are in addition to the standard schemes, such as North American Industry Classification System (NAICS), that are provided with the UDDI registry.

UDDI utility tools

You can use UDDI utility tools to import or export entities that use the UDDI Version 2 API.

UDDI user interface

The UDDI user console supports the UDDI Version 3 inquiry and publish APIs.

UDDI Version 3 client

The Java client for UDDI Version 3 handles the construction of raw SOAP requests for the client application. It is a JAX-RPC client and uses Version 3 data types, which are generated from the UDDI Version 3 Web Services Description Language (WSDL) and schema. These data types are serialized or deserialized to the XML, which constitutes the raw UDDI requests.

UDDI Version 2 clients

The following clients for UDDI Version 2 requests are provided:

- UDDI4J. A Java class library for issuing UDDI requests.

  Note: This client is provided in WebSphere Application Server Version 5 for both UDDI Version 1 requests (`uddi4j.jar`) and Version 2 requests (`uddi4jv2.jar`). These class libraries are still supported, as part of the `com.ibm.uddi.jar` file, but are deprecated in WebSphere Application Server Version 6.0.
- JAXR. The Java API for XML Registries (JAXR) is a Java client API for accessing UDDI and ebXML registries. WebSphere Application Server provides a JAXR provider for accessing the UDDI registry that conforms to the JAXR 1.0 specification.
- EJB. An Enterprise JavaBeans (EJB) interface for issuing UDDI Version 2 requests.

  Note: The UDDI EJB interface is still supported, but is deprecated in WebSphere Application Server Version 6.0.

Databases and production use of the UDDI registry

The UDDI registry fully supports a number of databases and can be used for development and test purposes. However, there are factors to consider when you decide which database is appropriate for your anticipated UDDI registry production use.

It is important to consult the information that is supplied by your chosen database vendor, but you must also consider the likely size and volume of requests, and whether the general performance and scalability of the UDDI registry is important.

For example, the Apache Derby database supports the full function of the UDDI registry, but it is not an enterprise level database and it does not have the same characteristics, for example, scaling or performance, as enterprise databases such as DB2.

Note: WebSphere Application Server supports direct customer use of the Apache Derby database in *test* environments only. The product does not support direct customer use of Apache Derby database in *production* environments.

If you need multiple connections to the UDDI registry database (for example to use the UDDI registry in a cluster configuration) and Apache Derby is your preferred database, you must use the network option for Apache Derby. This is because embedded Apache Derby has a limitation that allows only one Java virtual machine to access or load a database instance at any one time. That is, two application servers cannot access the same Apache Derby database instance at the same time.

Note: The UDDI registry can support multiple users, even if a single database connection exists.

UDDI registry terminology

Some terms specific to the UDDI registry are explained. Also, the relationship between the versions of the UDDI registry, the Organization for the Advancement of Structured Information (OASIS) specification, and the WebSphere Application Server level are shown.

IBM i Throughout the UDDI information in this information center, the directory locations of WebSphere Application Server are referred to as *app_server_root* and *profile_root*.

UDDI Definitions

bindingTemplate
AbindingTemplate is technical information about a service entry point and construction specifications.

businessEntity
A businessEntity is information about the party who publishes information about a family of services.

businessService
A businessService is descriptive information about a particular service.

customized UDDI node
A customized UDDI node is a UDDI node that is initialized with customized settings for the UDDI properties and UDDI policies. In particular, this type kind of node does not have default values for those properties that are read-only after initialization.

Use a customized UDDI node for anything other than simple testing purposes (for which a default UDDI node is enough). To set up a customized UDDI node, see the topic about setting up a customized UDDI node.

When you first start a customized UDDI node, you must set values for certain properties, and then initialize the node (using the administrative console or UDDI administrative interface), before the node is ready to accept UDDI requests. The properties that you must set control characteristics of the UDDI node that cannot be changed after initialization.

An advantage of using a customized UDDI node is that can set these properties to values that are suitable for your environment and usage of UDDI.

After a customized UDDI node is initialized, it is the same as a default UDDI node except that it uses customized UDDI property and policy values.

default UDDI node
A default UDDI node is a UDDI node that is initialized with default settings for the UDDI properties and UDDI policies, including the properties that are read-only after initialization. A default UDDI node is intended for use for testing and to provide a simple way to become familiar with the behavior of the UDDI registry.

You can set up a default UDDI node in two ways. The first is to run the uddiDeploy.jacl script, specifying the 'default' option, in which case the UDDI database will be an Apache Derby database that is created for you automatically.

IBM i The second is to create the database yourself, specifying the default option, which for Apache Derby is the DEFAULT parameter when using the UDDIDerbyCreate.jar file, and for DB2 or Oracle the SQL script insert_default_database_indicator.

After a default UDDI node is initialized, it is the same as a customized UDDI node except that it uses default UDDI property and policy values.

policy profile
A policy profile is set of UDDI policies. The default policy profile is the profile created when the default UDDI node is created. In the default policy profile, the nodeID and root key generator are set to read-only and cannot be changed after installation.

publisherAssertion
A publisherAssertion is information about a relationship between two parties, asserted by one or both.

tModel
A tModel (short for technical model) is a data structure representing a reusable concept, such as a web service type, a protocol used by web services, or a category system.

tModel keys in a service description are a technical “fingerprint” that you can use to trace the compatibility origins of a given service. They provide a common point of reference so that you can identify compatible services.

tModels are used to establish the existence of a variety of concepts and to point to their technical definitions. tModels that represent value sets such as category, identifier, and relationship systems are used to provide additional data to the UDDI core entities to aid discovery along a number of dimensions. This additional data is captured in keyedReferences that are in categoryBags, identifierBags, or publisherAssertions. The tModelKey attributes in these keyedReferences refer to the value set that relates to the concept or namespace being represented. The keyValues contain the values from that value set. In some cases, keyNames are significant, for example, to describe relationships and when using the general keywords value set. In all other cases, keyNames provide a version of the keyValue that people can read.

UDDI application
The UDDI application is the UDDI registry enterprise application.

UDDI entitlement
A UDDI entitlement is an entitlement that a UDDI user or publisher has in a UDDI registry, such as the capability to publish keyGenerators, or the tier to which the publisher is assigned (in other words, the number of entities that the publisher is entitled to publish). Each UDDI publisher has a range of settings for the various UDDI entitlements. A UDDI entitlement is sometimes referred to as a 'user entitlement', or as the UDDI publisher set of 'user entitlements'.

UDDI node
A UDDI node is a set of web services that supports at least one of the UDDI API sets, which supports interaction with UDDI data through the UDDI APIs. There is no direct mapping between a UDDI node and a WebSphere Application Server node. A UDDI node consists of an instance of the UDDI application running in an application server (or a cluster of UDDI application instances running in a cluster of application servers), together with an instance of the UDDI database containing UDDI data.

UDDI node initialization
UDDI node initialization is the process that sets up values in the UDDI database, and establishes the "personality" of the UDDI node. A UDDI node cannot accept UDDI API requests until it is initialized.

UDDI node state
The UDDI node state describes the current state of the UDDI node, as opposed to the state of the UDDI application (which is either stopped or started). A UDDI node can be in one of the following states:

- not initialized
- initialization pending
- initialization in progress

- migration pending
- migration in progress
- value set creation pending
- value set creation in progress
- activated
- deactivated

UDDI NodeId

The UDDI NodeId is a unique identifier of a UDDI node.

UDDI policy

A UDDI policy is a statement of required and expected behavior of a UDDI registry, specified by using policy values for the various policies that are defined in the UDDI Version specification.

UDDI property

A UDDI property is a value for a property that controls the personality or behavior of a UDDI node.

UDDI publisher

A UDDI publisher is a WebSphere Application Server user who is entitled to publish UDDI entities to a specified UDDI registry. A UDDI publisher is sometimes referred to as a 'UDDI user', or just as a 'publisher' when used in a UDDI context.

UDDI registry

A UDDI registry comprises one or more UDDI nodes. The UDDI registry in this version of WebSphere Application Server supports single-node UDDI registries only.

UDDI tier

A UDDI tier determines the number of UDDI entities of each type (business, services per business, bindings per service, tModel, publisher assertion) that a UDDI publisher is entitled to publish. Each UDDI publisher is assigned (either by default or explicitly by a UDDI administrator) to a particular tier, and cannot publish more entities than are allowed for that tier. There are some predefined tiers supplied with the UDDI registry, and a UDDI administrator can create additional tiers. A UDDI tier is often referred to just as a 'tier' when used in a UDDI context.

Version 2 UDDI registry

A Version 2 UDDI registry is a UDDI registry implementation that supports Version 2 of the UDDI specification and also Version 1. A Version 2 UDDI registry is included in WebSphere Application Server, Network Deployment Version 6.1.

Version 3 UDDI registry

A Version 3 UDDI registry is a UDDI registry implementation that supports Version 3 of the UDDI specification, and also Versions 1 and 2. A Version 3 UDDI registry is included in WebSphere Application Server. Note that Version 3 UDDI registry does not indicate a UDDI registry implementation that supports only UDDI Version 3 requests.

The following table shows how the various versions of the UDDI registry relate to the relevant OASIS specification and versions of WebSphere Application Server:

Table 69. UDDI registry versions and support. The table lists different UDDI registry versions, the associated OASIS UDDI specification levels supported and which versions of WebSphere Application Server support each UDDI registry.

UDDI registry Version	OASIS UDDI specification levels supported	Version of WebSphere Application Server that supports the UDDI registry
IBM i 3.0.2	• UDDI Version 1 • UDDI Version 2.0.4 (APIs), Version 2.0.3 (data structures) • UDDI Version 3.0.2	6.1 and later

Web Services Security concepts

The Web Services Security specification defines core facilities for protecting the integrity and confidentiality of a message, and provides mechanisms for associating security-related claims with a message.

What is new for securing web services

In WebSphere Application Server, there are many security enhancements for web services. The enhancements include supporting sections of the Web Services Security (WS-Security) specifications and providing architectural support for plugging in and extending the capabilities of security tokens.

Enhancements from the supported Web Services Security specifications

Since September 2002, the Organization for the Advancement of Structured Information Standards (OASIS) has been developing the Web Services Security (WS-Security) for SOAP message standard.

In April 2004, OASIS released the Web Services Security Version 1.0 specification, which is a major milestone for securing web services. In Feburary 2006, the specification was updated to Version 1.1. This specification is the foundation for other Web Services Security specifications and is also the basis for the Basic Security Profile (WS-I BSP) Version 1.0 specification, which was approved in March 2007.See the Basic Security Profile web page for more information.

Web Services Security Version 1.1 is a strategic move towards Web Services Security interoperability, and an important part of the Web Services Security roadmap. For more information on the Web Services Security roadmap, see Security in a Web Services World: A Proposed Architecture and Roadmap.

WebSphere Application Server supports the following OASIS specifications and WS-I profiles:

- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.1
- OASIS: Web Services Security: Kerberos Token Profile 1.1
- OASIS: WS-SecurityPolicy 1.2
- OASIS: WS-SecureConversation 1.3
- OASIS: WS-Trust 1.3
- Basic Security Profile (WS-I BSP) 1.0
- OASIS: Web Services Security: SAML Token Profile 1.1

The Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. Using SAML, a client can communicate assertions regarding the identity, attributes, and entitlements of a SOAP message. Using the SAML function in WebSphere Application Server, you can apply policy sets to JAX-WS applications to use SAML assertions in web services messages and in web services usage scenarios. Use SAML assertions to represent user identity and user security attributes, and optionally, to sign and to encrypt SOAP message elements.

For details on what parts of the previous specifications are supported in WebSphere Application Server, see “Supported functionality from OASIS specifications” on page 876.

High level features overview in WebSphere Application Server

In WebSphere Application Server, the Web Services Security for SOAP Message Version 1.1 specification is designed to be flexible and accommodate the requirements of Web services. For example, the specification does not have a mandatory security token definition. Instead, the specification defines a generic mechanism to associate the security token with a SOAP message. The use of security tokens is defined in the various Version 1.0 and 1.1 security token profiles, such as:

- The Username Token Profile

- The X.509 Token Profile
- The Kerberos Token Profile

For more information on security token profile development at OASIS, see Organization for the Advancement of Structured Information Standards.

The Web Services Security for SOAP Message Version 1.1 updates the Web Services Security for SOAP Message core specification and the various security token profiles. For this release, WebSphere Application Server implements the Username Token Profile 1.1 and the X.509 Token Profile 1.1, which includes support for the Thumbprint type of security token reference. In addition, it supports the signature confirmation and encrypted header portions of the Web Services Security Version 1.1 standard.

Important: The wire format (such as namespaces) in the WS-SecureConversation and WS-Trust 1.3 specification has changed. WebSphere Application Server tolerates requests formatted according to both the Submission Drafts and version 1.3 specifications, but you must ensure that the correct version is used when clients are communicating with a Web Services Feature Pack service provider. You can disable tolerance of the older format for WS-SecureConversation and WS-Trust 1.3 endpoints. Submission Drafts requests are not interoperable with version 1.3 standards.

WebSphere Application Server supports pluggable security tokens. The pluggable architecture is enhanced to support the Web Services Security specifications, other profiles, and other Web Services Security specifications. You can learn more about the pluggable security token framework for JAX-RPC web services, and associating custom security tokens with SOAP messages, by reading these articles on the IBM developerWorks website:

- Security for JAX-RPC Web services, Part 1: Generating custom tokens
- Security for JAX-RPC Web services, Part 2: Consuming custom tokens

WebSphere Application Server includes the following key enhancements:

- Support for the LTPA version 2 token
- Support for configuration of multiple callers, and an order attribute on the caller to determine which caller is used for the WebSphere credential
- Support for the published WS-SecurityPolicy version 1.2 specification embedded in WSDL
- Support for the WS-SecureConversation version 1.3 specification and the WS-Trust version 1.3 specification (used by WS-SecureConversation)
- Support for Kerberos token as defined in the WS-Kerberos Token Profile version 1.1 specification

For more information on some of these enhancements, see "Web Services Security enhancements" on page 872.

Configuration of Web Services Security

WebSphere Application Server uses the policy set model for implementing the Web Services Security Version 1.1 specification, including the Username token Version 1.1 profile, support for the Kerberos and LTPA v2 tokens, and the X.509 token version 1.1 profile. Policy sets combine configuration settings, including those for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, WS-SecureConversation, and WS-Security. For more information on policy sets, refer to the topic Managing policy sets using the administrative console.

You can use the administrative console to configure the Web Services Security binding of a deployed application with Web Services Security constraints that are defined in the policy set.

For the X.509 Certificate Token Profile, one new type of security token reference is the Thumbprint reference, which is specified in the binding. WebSphere Application Server now supports creating and authenticating a security token by using a security token reference (STR) with a key identifier and a Thumbprint in the <KeyInfo> element. The Thumbprint key information type requires that there be a keystore with the public and private key pair instead of a shared key. To use the Thumbprint of the specified certificate, specify the keyInfo type THUMBPRINT in the bindings.

For example, a decryption key is referenced by means of the thumbprint of an associated certificate. The certificate is not included in the message. Instead, the <ds:KeyInfo> element contains a <wsse:SecurityTokenReference> element that specified the thumbprint of the specified certificate by means of the http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1 attribute of the <wsse:KeyIdentifier> element.

To take advantage of implementations associated with the Web Services Security Version 1.1 specification, you must:

- Ensure that your applications use the Java API for XML Web Services (JAX-WS) programming model.
- Re-configure the Web Services Security constraints in the new policy set and binding format.

WebSphere Application Server provides the following tools that you can use to edit the policy set file and the binding file:

IBM assembly tools

You can use IBM assembly tools to develop web services and configure the policy set and the binding file for Web Services Security. The tools enable you to assemble both web and Enterprise JavaBeans (EJB) modules. The assembly tools do not support direct editing of policy sets, but can import policy sets from the application server, and then attach the modified policy sets to the service. For more information, read about assembly tools.

Note: You can use policy sets only with Java API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

WebSphere Application Server administrative console

You can use the administrative console to configure the Web Services Security binding of a deployed application with Web Services Security constraints that are defined in the policy set.

What is not supported

Web service security is still fairly new and some of the standards are still being defined or standardized. The following functionality is not supported in WebSphere Application Server:

- JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification). See the standard documentation for more information: JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification).
- Application programming interfaces (API) do not exist for Web Services Security in WebSphere Application Server Versions 6.0.x and later.
- SAML token profile is not supported out of the box.
- REL token profile is not supported.
- SwA profile is not supported

What is supported by the IBM Software Development Kit (SDK)

The following standards exist for the Java application programming interface for XML security and Web Services Security:

- JSR-105 (Java API for XML-Signature XPath Filter Version 2.0

  W3C Recommendation, November 2002
- JSR-106 (Java API for XML Encryption Syntax and Processing)

W3C Recommendation, December 2002

For more information on the IBM SDK for Java Version 6, see the security information documentation.

For information on what is supported for Web Services Security in WebSphere Application Server, see "Supported functionality from OASIS specifications" on page 876.

Web Services Security enhancements

WebSphere Application Server includes a number of enhancements for securing web services. For example, policy sets are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later, to simplify security configuration for web services.

Building your applications

The Web Services Security runtime implementation used by WebSphere Application Server Version 8 is based on the Java API for XML Web Services (JAX-WS) programming model. The JAX-WS runtime environment is based on Apache Open Source Axis2, and the data model is AXIOM. Instead of deployment descriptor and bindings, a policy set is used for configuration. You can use the WebSphere Application Server administrative console to edit the application binding files associated with the policy sets. The JAX-WS runtime environment is supported for the WebSphere Application Server V6.1 Feature Pack for Web Services, and later.

The JAX-RPC programming model, which uses deployment descriptors and bindings, is still supported. Read the topic Securing JAX-RPC Web services using message level security for more information.

Using policy sets

Use policy sets to simplify your web service Quality of Service configuration.

Note: Policy sets can only be used with JAX-WS applications, in WebSphere Application Server V6.1 Feature Pack for Web Services, and later. Policy sets cannot be used for JAX-RPC applications.

Policy sets combine configuration settings, including those for transport and message level configuration, such as Web Services Addressing (WS-Addressing), Web Services Reliable Messaging (WS-ReliableMessaging), and Web Services Security (WS-Security), which includes Secure Conversation (WS-SecureConversation).

Managing trust policies

Web Services Security Trust (WS-Trust) provides the ability for an endpoint to issue a security context token for Web Services Secure Conversation (WS-SecureConversation). The token issuing support is limited to the security context token. Trust policy management defines a policy for each of the trust service operations, such as issuing, cancelling, validating, and renewing a token. A client's bootstrap policies must correspond to the WebSphere Application Server trust service policies.

Securing session-based messages

Web Services Secure Conversation provides a secured session for long running message exchanges and leveraging symmetric cryptographic algorithm. WS-SecureConversation provides the basic security for securing session-based messages exchange patterns, such as Web Services Security Reliable Messaging (WS-ReliableMessaging).

Updating message-level security

Web Services Security (WS-Security) Version 1.1 supports the following functions that update the message-level security.

- Signature confirmation
- Encrypted headers

Signature confirmation enhances the protection of XML digital signature security. The <SignatureConfirmation> element indicates that the responder has processed the signature in the request, and the signature confirmation ensures that the signature is indeed processed by the intended recipient. To process signature confirmation correctly, the initiator must preserve the signatures during the request generation processing and later must retrieve the signatures for confirmation checks even with the stateless nature of web services and the different message exchange patterns. You enable signature confirmation by configuring the policy.

The encrypted header element provides a standard way of encrypting SOAP headers, which helps inter-operability. As defined in the SOAP message security specification, the <EncryptedHeader> element indicates that a specific SOAP header (or set of headers) must be protected. Encrypting SOAP headers and parts helps to provide more secure message-level security. The EncryptedHeader element ensures compliance with the SOAP mustUnderstand processing guidelines and prevents disclosure of information contained in attributes on a SOAP header block.

Using identity assertion

In a secured environment such as an intranet, a secure sockets layer (SSL) connection or through a Virtual Private Network (VPN), it is useful to send the requester identity only without credentials, such as password, with other trusted credentials, such as the server identity. WebSphere Application Server supports the following types of identity assertions:

- A username token without a password
- An X.509 Token for a X.509 certificate

For more information about identity assertion, read the topic Trusted ID evaluator.

Signing or encrypting data with a custom token

For the JAX-RPC programming model, the key locator, or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface, is enhanced to support the flexibility of the specification. The key locator is responsible for locating the key. The local JAAS Subject is passed into the KeyLocator.getKey() method in the context. The key locator implementation can derive the key from the token, which is created by the token generator or the token consumer, to sign a message, to verify the signature within a message, to encrypt a message, or to decrypt a message. The com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface is different from the version in WebSphere Application Server Version 5.x. The com.ibm.wsspi.wssecurity.config.KeyLocator interface from Version 5.x is deprecated. There is no automatic migration for the key locator from Version 5.x to Versions 6 and later. You must migrate the source code for the Version 5.x key locator implementation to the key locator programming model for Version 6 and later.

For the JAX-WS programming model, the pluggable token framework reuses the same framework from the WSS API. The same implementation for creating and validating a security token can be used in both the Web Services Security run time and the WSS API application. This simplifies the SPI programming model and makes it easier to add new or custom security token types. The redesigned SPI consists of the following interfaces:

- The JAAS CallbackHandler and JAAS Login Module create security tokens on the generator side and validate, or authenticate, security tokens on the consumer side.
- The Security Token interface, com.ibm.websphere.wssecurity.wssapi.token.SecurityToken, represents the security token that has methods to get the identity, XML format and cryptographic keys.

When using JAX-WS, the following interfaces are no longer required:

- Token Generator (com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent)

- Token Consumer (com.ibm.wsspi.wssecurity.token.TokenConsumerComponent)
- Key Locator (com.ibm.wsspi.wssecurity.keyinfo.KeyLocator)

You can learn more about custom security tokens by reading these articles on the IBM developerWorks website:

- Security for JAX-RPC Web services, Part 1: Generating custom tokens
- Security for JAX-RPC Web services, Part 2: Consuming custom tokens

Signing or encrypting any XML element

An XPath expression is used for selecting which XML element to sign or encrypt. However, an envelope signature is used when you sign the SOAP envelope, SOAP header, or Web Services Security header. In JAX-RPC web services, the XPath expression is specified in the application deployment descriptor. In JAX-WS web services, the XPath expression is specified in the WS-Security policy of the policy set.

The JAX-WS programming model uses policy sets to indicate the message parts where security should be applied. For example, the <Body> assertion is used to indicate that the body of the SOAP message is signed or encrypted. Another example is the <Header> assertion, where the QName of the SOAP header to be signed or encrypted is specified.

Signing or encrypting SOAP headers

The OASIS Web Services Security (WS-Security) Version 1.1 support provides for a standard way of encrypting and signing SOAP headers. To sign or encrypt SOAP messages, specify the QName to select header elements in the SOAP header of the SOAP message.

You can configure policy sets for signing or encrypting either by using the administrative console or by using Web Services Security APIs (WSS APIs). For more details, see the topic Securing message parts using the administrative console.

For signing, specify the following:

Name This optional attribute indicates the local name of the SOAP header to be integrity protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

Namespace
This required attribute indicates the namespace of the SOAP headers to be integrity protected.

For encrypting, specify the following:

Name This optional attribute indicates the local name of the SOAP header to be confidentiality protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

Namespace
This required attribute indicates the namespace of the SOAP header(s) to be confidentiality protected.

This results in an <EncryptedHeader> element that contains the <EncryptedData> element.

For Web Services Security Version 1.0 behavior, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.0 property with a value of `true` in EncryptionInfo in the bindings. Specifying this property results in an <EncryptedData> element.

For Web Services Security Version 1.1 behavior that is equivalent to WebSphere Application Server versions prior to version 7.0, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.1.pre.V7 property with a value of `true` on the <encryptionInfo> element in the binding. When this property is specified, the <EncryptedHeader> element includes a wsu:Id parameter and the <EncryptedData> element omits the Id parameter. This property should only be used if compliance with Basic Security Profile 1.1 is not required and it is necessary to send <EncryptedHeader> elements to a client or server that uses the WebSphere Application Server Version 5.1 Feature Pack for Web Services.

Supporting LTPA

Lightweight Third Party Authentication (LTPA) is supported as a binary security token in Web Services Security. Web Services Security supports both LTPA (version 1) and LTPA version 2 tokens. The LTPA version 2 token, which is more secure than version 1, is supported in WebSphere Application Server version 7.0 and later.

Extending the support for timestamps

You can insert a timestamp in other elements during the signing process besides the Web Services Security header. This timestamp provides a mechanism for adding a time limit to an element. This support is an extension for WebSphere Application Server. Other vendor implementations might not have the ability to consume a message that is generated with an additional timestamp that is inserted in the message.

Extending the support for nonce

You can insert a nonce, which is a randomly generated value, in other elements beside the Username token. The nonce is used to reduce the chance of a replay attack. This support is an extension for WebSphere Application Server. Other vendor implementations might not have the ability to consume messages with a nonce that is inserted into elements other than a Username token.

Supporting distributed nonce caching

Distributed nonce caching is a new feature for web services in WebSphere Application Server Versions 6 and later that enables you to replicate nonce data between servers in a cluster. For example, you might have application server A and application server B in cluster C. If application server A accepts a nonce with a value of `X`, then application server B creates a SoapSecurityException if it receives the nonce with the same value within a specified period of time.

Caching the X.509 certificate

WebSphere Application Server caches the X.509 certificates it receives, by default, to avoid certificate path validation and improve its performance. However, this change might lead to security exposure. You can disable X.509 certificate caching by using the following steps:

On the server level:

- Click **Servers > Application servers > *server_name*** .
- Under Security, click **Web services: Default bindings for Web Services Security**.
- Under Additional properties, click **Properties > New**.
- In the Property name field, type `com.ibm.ws.wssecurity.config.token.certificate.useCache`.
- In the Property value field, type `false`.

Providing support for a certificate revocation list

The certificate revocation list (CRL) in WebSphere Application Server is used to enhance certificate path validation. You can specify a CRL in the collection certificate store for validation. You can also encode a CRL in an X.509 token using PKCS#7 encoding. However, WebSphere Application Server Version 6 and later do not support X509PKIPathv1 CRL encoding in a X.509 token.

Important: The PKCS#7 encoding was tested with the IBM certificate path (IBM CertPath) provider only. The encoding is not supported for other certificate path providers.

Supported functionality from OASIS specifications

The application server supports the Organization for the Advancement of Structured Information (OASIS) Web Services Security (WS-Security) specifications.

WebSphere Application Server supports these OASIS Web Services Security Version 1.0 specifications.

- OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.0
- OASIS: Web Services Security X.509 Certificate Token Profile 1.0

In WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later, support for the OASIS standards has been updated to the latest versions of Web Services Security (WS-Security) specifications and tokens. Web Services Security Version 1.1 provides better security verification for signature, a standard way of encrypting SOAP headers, and meets the requirement from some of the inter-operability scenarios that use features from Web Services Security Version 1.1.

- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security UsernameToken Profile 1.1 (Standard Specification, 1 February 2006)
- OASIS: Web Services Security X.509 Certificate Token Profile 1.1 (Standard Specification, 1 February 2006)

The following standards are supported only in WebSphere Application Server Version 7.0 and later.

- WS-Security Kerberos Token Profile 1.1
- WS-SecureConversation Version 1.3
- WS-Trust Version 1.3
- WS-SecurityPolicy Version 1.2

WS-SecurityPolicy support is only available for Web Services Metadata Exchange (WS-MetadataExchange) scenarios where the assertions are embedded in the WSDL file. For more information, read the WS-MetadataExchange requests topic.

In 2007, the OASIS Web Services Secure Exchange Technical Committee (WS-SX) produced and approved the following specifications. Portions of these specifications are supported by WebSphere Application Server Version 7 and later.

- WS-SecureConversation
- WS-Trust
- WS-SecurityPolicy

OASIS: Web Services Security SOAP Message Security 1.0 and 1.1

The following table shows the aspects of the OASIS: Web Services Security: SOAP Message Security 1.0 and 1.1 specifications that are supported in WebSphere Application Server Versions 6 and later.

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Security header	• @S11:actor (for an intermediary) • @S11:mustUnderstand • @S12:mustUnderstand • @S12:role (S12 is the namespace prefix for http://www.w3.org/2003/05/soap-envelope when using SOAP Version 1.2)

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	**Specific aspect that is supported**
Security tokens	• Username token (user name and password) • Binary security token (X.509 and Lightweight Third Party Authentication (LTPA) • Custom token – Other binary security token – XML token **Note:** WebSphere Application Server does not provide an implementation, but you can use an XML token with plug-in point.
Token references	• Direct reference • Key identifier • Key name • Embedded reference
Signature	Signature confirmation

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Signature algorithms	• **Digest** **SHA1** http://www.w3.org/2000/09/xmldsig#sha1 **SHA256** http://www.w3.org/2001/04/xmlenc#sha256 **SHA512** http://www.w3.org/2001/04/xmlenc#sha512 • **MAC** **HMAC-SHA1** http://www.w3.org/2000/09/xmldsig#hmac-sha1 • **Signature** **DSA with SHA1** http://www.w3.org/2000/09/xmldsig#dsa-sha1 Do not use this algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP) **RSA with SHA1** http://www.w3.org/2000/09/xmldsig#rsa-sha1 • **Canonicalization** **Canonical XML (with comments)** http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments **Canonical XML (without comments)** http://www.w3.org/TR/2001/REC-xml-c14n-20010315 **Exclusive XML canonicalization (with comments)** http://www.w3.org/2001/10/xml-exc-c14n#WithComments **Exclusive XML canonicalization (without comments)** http://www.w3.org/2001/10/xml-exc-c14n# • **Transform** **STR transform** http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage- security-1.0#STR-Transform **XPath** http://www.w3.org/TR/1999/REC-xpath-19991116 Do not use the original XPATH transform if you want your configured application to be in compliance with the Basic Security Profile (BSP). **Note:** When referring to an element in a SECURE_ENVELOPE that does not carry an attribute of type ID from a ds:Reference in a SIGNATURE, you must use the XPATH Filter 2.0 Transform, http://www.w3.org/2002/06/xmldsig-filter2 **Enveloped signature** http://www.w3.org/2000/09/xmldsig#enveloped-signature **XPath Filter2** http://www.w3.org/2002/06/xmldsig-filter2 **Note:** When referring to an element in a SECURE_ENVELOPE that does not carry an ID attribute type from a ds:Reference in a SIGNATURE, you must use the XPATH Filter 2.0 Transform, http://www.w3.org/2002/06/xmldsig-filter2 **Decryption transform** http://www.w3.org/2002/07/decrypt#XML

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Signature signed parts for JAX-RPC only	• WebSphere Application Server key words: – body, which signs the SOAP message body – timestamp, which signs all of the time stamps – securitytoken, which signs all of the security tokens – dsigkey, which signs the signing key – enckey, which signs the encryption key – messageid, which signs the `wsa :MessageID` element in WS-Addressing. – to, which signs the `wsa:To` element in WS-Addressing – action, which signs the `wsa:Action` element in WS-Addressing – relatesto, which signs the `wsa:RelatesTo` element in WS-Addressing wsa is the namespace prefix of http://schemas.xmlsoap.org/ws/2004/08/addressing – wscontext, which specifies the WS-Context header for the SOAP header. – wsafrom, which specifies the `<wsa:From>` WS-Addressing From element in the SOAP header. – wsareplyto, which specifies the `<wsa:ReplyTo>` WS-Addressing ReplyTo element in the SOAP header. – wsafaultto, which specifies the `<wsa:FaultTo>` WS-Addressing FaultTo element in the SOAP header. – wsaall, which specifies all of the WS-Addressing elements in the SOAP header. • XPath expression to select an XML element in a SOAP message. For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Signature message parts for JAX-WS only	• Body (which signs the SOAP message body) • Header (which signs one or more SOAP headers within the main SOAP header) • XPath expression to select an XML element in a SOAP message. – For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Encryption	EncryptedHeader element

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Encryption algorithms	**Important:** Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country, its regulations, and its policies concerning the import, possession, use, and re-export of encryption software, to determine if it is permitted. • Data encryption – Triple DES in CBC: http://www.w3.org/2001/04/xmlenc#tripledes-cbc – AES128 in CBC: http://www.w3.org/2001/04/xmlenc#aes128-cbc – AES192 in CBC: http://www.w3.org/2001/04/xmlenc#aes192-cbc This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the Encryption information configuration settings: Message parts. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). – AES256 in CBC: http://www.w3.org/2001/04/xmlenc#aes256-cbc This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the Encryption information configuration settings: Message parts. • Key encryption – Key transport (public key cryptography) - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p. **Note:** • When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5. • Use of the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine does not support this transport algorithm. - RSA Version 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5 – Symmetric key wrap (private key cryptography) - Triple DES key wrap: http://www.w3.org/2001/04/xmlenc#kw-tripledes - AES key wrap (aes128): http://www.w3.org/2001/04/xmlenc#kw-aes128 - AES key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192 This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the Encryption information configuration settings: Message parts. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). - AES key wrap (aes256): http://www.w3.org/2001/04/xmlenc#kw-aes256 This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the Encryption information configuration settings: Message parts. • Manifests-xenc is the namespace prefix of http://www.w3.org/TR/xmlenc-core – xenc:ReferenceList – xenc:EncryptedKey Advanced Encryption Standard (AES) is designed to provide stronger and better performance for symmetric key encryption over Triple-DES (data encryption standard). Therefore, it is recommended that you use AES, if possible, for symmetric key encryption.
Encryption message parts for JAX-RPC only	• WebSphere Application Server keywords – bodycontent, which is used to encrypt the SOAP body content – usernametoken, which is used to encrypt the username token – digestvalue, which is used to encrypt the digest value of the digital signature – signature, which is used to encrypt the entire digital signature – wscontextcontent, which encrypts the content in the WS-Context header for the SOAP header. • XPath expression to select the XML element in the SOAP message – XML elements – XML element contents

Table 70. Aspects of OASIS SOAP Message Security standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Encryption message parts for JAX-WS only	• Body (which encrypts the SOAP message body content) • Header (which encrypts one or more SOAP headers within the main SOAP header, resulting in the EncryptedHeader element) • XPath expression to select an XML element in a SOAP message – For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Time stamp	• Within Web Services Security header • WebSphere Application Server is extended to allow you to insert time stamps into other elements so that the age of those elements can be determined.
Error handling	SOAP faults • New `failure` SOAP fault with faultcode • `The message has expired` text has been added

OASIS: Web Services Security UsernameToken Profile 1.0

The following table shows the aspects of the OASIS: Web Services Security Username Token Profile 1.0 specification that is supported in WebSphere Application Server.

Table 71. Aspects of OASIS Username Token Profile V1.0 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Password types	Text
Token references	Direct reference

OASIS: Web Services Security UsernameToken Profile 1.1

The following table shows the aspects of the OASIS: Web Services Security Username Token Profile 1.1 specification that is supported in WebSphere Application Server. Items that were previously supported for Web Services Security UsernameToken Profile 1.0 are not listed but are still supported, unless noted otherwise.

Table 72. Aspects of OASIS Username Token Profile V1.1 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Password types	Text
Token references	Direct reference

OASIS: Web Services Security X.509 Certificate Token Profile 1.0

The following table shows the aspects of the OASIS: Web Services Security X.509 Certificate Token Profile specification that are supported in WebSphere Application Server Versions 6 and later.

Table 73. Aspects of OASIS X.509 Certificate Token V1.0 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Token types	• X.509 Version 3: Single certificate http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509v3 • X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL) http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509PKIPathv1 • X.509 Version 3: PKCS7 with or without CRLs. The IBM software development kit (SDK) supports both. The Sun Java SE Development Kit 6 (JDK 6) supports PKCS7 without CRL only.

Table 73. Aspects of OASIS X.509 Certificate Token V1.0 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Token references	• Key identifier – subject key identifier • Direct reference • Custom reference – issuer name and serial number

OASIS: Web Services Security X.509 Certificate Token Profile 1.1

The following table shows the aspects of the OASIS: Web Services Security X.509 Certificate Token Profile 1.1 specification that are supported in WebSphere Application Server. Items that were previously supported for Web Services Security X.509 Certificate Token Profile 1.0 are not listed but are still supported, unless noted otherwise.

Table 74. Aspects of OASIS X.509 Certificate Token V1.1 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Token types	X.509 Version 1: Single certificate
Token references	Key identifier – subject key identifier • Can only reference an X.509v3 certificate • Can specify the thumbprint of the specified certificate by using the http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1 attribute of the <wsse:KeyIdentifier> element.

OASIS: Web Services Security Kerberos Token Profile 1.1

The following table shows the aspects of the OASIS: Web Services Security Kerberos Token Profile 1.1 specification that are supported in WebSphere Application Server.

Table 75. Aspects of OASIS Kerberos Token Profile standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Token types	• GSS_API Kerberos v5 token http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ • GSS_API Kerberos v5 token per RFC1510 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510 • GSS_API Kerberos v5 token per RFC4120 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120 • Kerberos v5 token http://docs.oasis-open.org/wss/oasiswss- kerberos-token-profile-1.1#Kerberosv5_AP_REQ • Kerberos v5 token per RFC1510 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510 • Kerberos v5 token per RFC4120 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ412
Token references	• Security token reference • Key identifier, which is used after the initial Kerberos v5 token is consumed • Derived key token based on the Kerberos key

OASIS: Web Services Security WS-Secure Conversation Draft and Version 1.3

The following table shows the aspects of the OASIS: WS-SecureConversation specification that are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later. Support for Version 1.3 of the specification is provided in WebSphere Application Server Version 7.0 and later.

Table 76. Aspects of OASIS SecureConversation standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Token types	• Security Context Token draft version: http://schemas.xmlsoap.org/ws/2005/02/sc/sct • Security Context Token Version 1.3: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
Token references	Direct reference
Security context establishment	Security context token created by a security token service that is embedded in the WebSphere Application Server.
Renewing context	Automatic renewal of the token when its about to expire.
Cancelling context	Explicit cancel request support.
Derived keys	The following information is used to derive the keys using a shared secret from a security context: • /wsc:DerivedKeyToken/wsse:SecurityTokenReference • /wsc:DerivedKeyToken/wsc:Label • /wsc:DerivedKeyToken/wsc:Nonce • /wsc:DerivedKeyToken/wsc:Length
Error handling	SOAP faults, including: • wsc:BadContextToken • wsc:UnsupportedContextToken • wsc:RenewNeeded • wsc:UnableToRenew

OASIS: Web Services Security WS-Trust Version 1.0 Draft and Version 1.3

The following tables show the aspects of the OASIS: Web Services Security: WS-Trust Version 1.0 Draft and Version 1.3 specifications that are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later.

Table 77. Aspects of OASIS Trust V1.0 and V1.3 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Namespace	http://schemas.xmlsoap.org/ws/2005/02/trust
Request header	/wsa:Action Valid options include: • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Renew • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Cancel • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Validate

Table 77. Aspects of OASIS Trust V1.0 and V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Request elements and attributes	/wst:RequestSecurityToken /wst:RequestSecurityToken/@Context /wst:RequestSecurityToken/wst:RequestType • Valid options include: – http://schemas.xmlsoap.org/ws/2005/02/trust/Issue – http://schemas.xmlsoap.org/ws/2005/02/trust/Renew – http://schemas.xmlsoap.org/ws/2005/02/trust/Cancel – http://schemas.xmlsoap.org/ws/2005/02/trust/Validate /wst:RequestSecurityToken/wst:TokenType • Valid options include: – for http://schemas.xmlsoap.org/ws/2005/02/sc/sct - /wst:RequestSecurityToken/wsp:AppliesTo - /wst:RequestSecurityToken/wst:Entropy - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type – for http://schemas.xmlsoap.org/ws/2005/02/trust/Nonce - /wst:RequestSecurityToken/wst:Lifetime - /wst:RequestSecurityToken/wst:Lifetime/wsu:Created - /wst:RequestSecurityToken/wst:Lifetime/wsu:Expires - /wst:RequestSecurityToken/wst:KeySize - /wst:RequestSecurityToken/wst:KeyType – for http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey - /wst:RequestSecurityToken/wst:RenewTarget - /wst:RequestSecurityToken/wst:Renewing - /wst:RequestSecurityToken/wst:Renewing/@Allow - /wst:RequestSecurityToken/wst:Renewing/@OK - /wst:RequestSecurityToken/wst:CancelTarget - /wst:RequestSecurityToken/wst:ValidateTarget - /wst:RequestSecurityToken/wst:Issuer
Response header	/wsa:Action Valid options include: • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Renew • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Cancel • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Validate

Table 77. Aspects of OASIS Trust V1.0 and V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Response elements and attributes	/wst:RequestSecurityTokenResponse /wst:RequestSecurityTokenResponse/@Context /wst:RequestSecurityTokenResponse/wst:TokenType /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wsp:AppliesTo /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wst:RequestedAttachedReference /wst:RequestSecurityTokenResponse/wst:RequestedUnattachedReference /wst:RequestSecurityTokenResponse/wst:RequestedProofToken /wst:RequestSecurityTokenResponse/wst:Entropy /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret/@Type /wst:RequestSecurityTokenResponse/wst:Lifetime /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Created /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Expires /wst:RequestSecurityTokenResponse/wst:RequestedProofToken/wst:ComputedKey /wst:RequestSecurityTokenResponse/wst:KeySize /wst:RequestSecurityTokenResponse/wst:Renewing /wst:RequestSecurityTokenResponse/wst:Renewing/@Allow /wst:RequestSecurityTokenResponse/wst:Renewing/@OK /wst:RequestSecurityTokenResponse/wst:RequestedTokenCancelled /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status/wst:Code • Valid responses include: – http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid – http://schemas.xmlsoap.org/ws/2005/02/trust/status/invalid /wst:RequestSecurityTokenResponse/wst:Status/wst:Reason

Table 77. Aspects of OASIS Trust V1.0 and V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Error handling	wst:InvalidRequest wst:FailedAuthentication wst:RequestFailed wst:InvalidSecurityToken wst:AuthenticationBadElements wst:BadRequest wst:ExpiredData wst:InvalidTimeRange wst:InvalidScope wst:RenewNeeded wst:UnableToRenew

Table 78. Aspects of OASIS Trust V1.3 standard supported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Namespace	http://docs.oasis-open.org/ws-sx/ws-trust/200512
Request header	/wsa:Action Valid options include: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Renew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Cancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Validate • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchIssue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchCancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchRenew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchValidate

Table 78. Aspects of OASIS Trust V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Request elements and attributes	/wst:RequestSecurityToken /wst:RequestSecurityToken/@Context /wst:RequestSecurityToken/wst:RequestType • Valid options include: – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchIssue – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchRenew – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchCancel – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchValidate /wst:RequestSecurityToken/wst:TokenType • Valid options include: – for http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct - /wst:RequestSecurityToken/wsp:AppliesTo - /wst:RequestSecurityToken/wst:Entropy - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type – for http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce - /wst:RequestSecurityToken/wst:Lifetime - /wst:RequestSecurityToken/wst:Lifetime/wsu:Created - /wst:RequestSecurityToken/wst:Lifetime/wsu:Expires - /wst:RequestSecurityToken/wst:KeySize - /wst:RequestSecurityToken/wst:KeyType – for http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey - /wst:RequestSecurityToken/wst:RenewTarget - /wst:RequestSecurityToken/wst:Renewing - /wst:RequestSecurityToken/wst:Renewing/@Allow - /wst:RequestSecurityToken/wst:Renewing/@OK - /wst:RequestSecurityToken/wst:CancelTarget - /wst:RequestSecurityToken/wst:ValidateTarget - /wst:RequestSecurityToken/wst:Issuer
Response header	/wsa:Action Valid options include: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/RenewFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/CancelFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/ValidateFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/CancelFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/RenewFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/ValidateFinal

Table 78. Aspects of OASIS Trust V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Response elements and attributes	/wst:RequestSecurityTokenResponse /wst:RequestSecurityTokenResponse/@Context /wst:RequestSecurityTokenResponse/wst:TokenType /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wsp:AppliesTo /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wst:RequestedAttachedReference /wst:RequestSecurityTokenResponse/wst:RequestedUnattachedReference /wst:RequestSecurityTokenResponse/wst:RequestedProofToken /wst:RequestSecurityTokenResponse/wst:Entropy /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret/@Type /wst:RequestSecurityTokenResponse/wst:Lifetime /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Created /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Expires /wst:RequestSecurityTokenResponse/wst:RequestedProofToken/wst:ComputedKey /wst:RequestSecurityTokenResponse/wst:KeySize /wst:RequestSecurityTokenResponse/wst:Renewing /wst:RequestSecurityTokenResponse/wst:Renewing/@Allow /wst:RequestSecurityTokenResponse/wst:Renewing/@OK /wst:RequestSecurityTokenResponse/wst:RequestedTokenCancelled /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status/wst:Code • Valid responses include: – http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid – http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/invalid /wst:RequestSecurityTokenResponse/wst:Status/wst:Reason

Table 78. Aspects of OASIS Trust V1.3 standard supported in WebSphere Application Server (continued). Use the table to determine which aspects of the OASIS standard are supported.

Supported topic	Specific aspect that is supported
Error handling	wst:InvalidRequest wst:FailedAuthentication wst:RequestFailed wst:InvalidSecurityToken wst:AuthenticationBadElements wst:BadRequest wst:ExpiredData wst:InvalidTimeRange wst:InvalidScope wst:RenewNeeded wst:UnableToRenew

Functionality that is not supported by WebSphere Application Server

The following list shows the functionality that is supported in the OASIS specifications, OASIS drafts, and other recommendations but is not supported by WebSphere Application Server Version 6 and later:

- Web Services Security SOAP Messages with Attachments (SwA) profile 1.0

  Note: When using the JAX-WS programming model, securing the SOAP Message Transmission Optimization Mechanism (MTOM) attachment is supported. See the topic Enabling MTOM for JAX-WS web services for more information.
- XrML token profile
- XML enveloping digital signature
- XML enveloping digital encryption
- The following WS-SecureConversation functionality is not supported by WebSphere Application Server:
  - Two methods for establishing security context are not supported: 1) security context token created by one of the communicating parties and propagated with a message; and 2) security context token created through negotiation or exchanges.
  - SCT propagation
  - Amending security contexts
- The following transform algorithms for digital signatures are not supported:
  - XSLT: http://www.w3.org/TR/1999/REC-xslt-19991116
  - SOAP Message Normalization

    See SOAP Version 1.2 Message Normalization for information, such as an empty header or header entry with `mustUnderstand=false` is removed, and so forth.
  - Decryption transform
- The following key agreement algorithm for encryption is not supported:
  - Diffie-Hellman: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html#sec-DHKeyValue
- The following canonicalization algorithm for encryption, which is optional in the XML encryption specification, is not supported:
  - Canonical XML with or without comments
  - Exclusive XML Canonicalization with or without comments

- DSA digital signature is not supported.
- Pre-agreed symmetric key data encryption is not supported.
- Auditing for nonrepudiation for digital signatures is not supported.
- In both versions of the Username Token Profile specification, the digest password type is not supported.
- In the Username Token Version 1.1 Profile specification, the key derivation based on a password is not supported.

Unsupported function for WS-Trust Version 1.0 Draft and Version 1.3

The following tables show the aspects of the OASIS: Web Services Security: WS-Trust Version 1.0 Draft and Version 1.3 specifications that are **not** supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later.

Table 79. Aspects of OASIS Trust V1.0 and V1.3 standard that are unsupported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are not supported.

Unsupported topic	Specific aspect that is not supported
Elements and attributes	/wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type Unsupported request options: • for http://schemas.xmlsoap.org/ws/2005/02/trust/AsymmetricKey and http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey – /wst:RequestSecurityToken/wst:Claims – /wst:RequestSecurityToken/wst:AllowPostdating – /wst:RequestSecurityToken/wst:OnBehalfOf – /wst:RequestSecurityToken/wst:AuthenticationType – /wst:RequestSecurityToken/wst:KeyType • for http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey – /wst:RequestSecurityToken/wst:SignatureAlgorithm – /wst:RequestSecurityToken/wst:EncryptionAlgorithm – /wst:RequestSecurityToken/wst:CanonicalizationAlgorithm – /wst:RequestSecurityToken/wst:ComputedKeyAlgorithm – /wst:RequestSecurityToken/wst:Encryption – /wst:RequestSecurityToken/wst:ProofEncryption – /wst:RequestSecurityToken/wst:UseKey – /wst:RequestSecurityToken/wst:UseKey/@Sig – /wst:RequestSecurityToken/wst:SignWith – /wst:RequestSecurityToken/wst:EncryptWith – /wst:RequestSecurityToken/wst:DelegateTo – /wst:RequestSecurityToken/wst:Forwardable – /wst:RequestSecurityToken/wst:Delegatable – /wst:RequestSecurityToken/wsp:Policy – /wst:RequestSecurityToken/wsp:PolicyReference
Response elements and attributes	/wst:RequestSecurityTokenResponseCollection /wst:RequestSecurityTokenResponseCollection/wst:RequestSecurityTokenResponse

Table 80. Aspects of OASIS Trust V1.3 standard that are unsupported in WebSphere Application Server. Use the table to determine which aspects of the OASIS standard are not supported.

Unsupported topic	Specific aspect that is not supported
Elements and attributes	/wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type Unsupported request options: • for http://docs.oasis-open.org/ws-sx/ws-trust/200512/AsymmetricKey and http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey – /wst:RequestSecurityToken/wst:Claims – /wst:RequestSecurityToken/wst:AllowPostdating – /wst:RequestSecurityToken/wst:OnBehalfOf – /wst:RequestSecurityToken/wst:AuthenticationType – /wst:RequestSecurityToken/wst:KeyType • for http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey and http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer – /wst:RequestSecurityToken/wst:SignatureAlgorithm – /wst:RequestSecurityToken/wst:EncryptionAlgorithm – /wst:RequestSecurityToken/wst:CanonicalizationAlgorithm – /wst:RequestSecurityToken/wst:ComputedKeyAlgorithm – /wst:RequestSecurityToken/wst:Encryption – /wst:RequestSecurityToken/wst:ProofEncryption – /wst:RequestSecurityToken/wst:UseKey – /wst:RequestSecurityToken/wst:UseKey/@Sig – /wst:RequestSecurityToken/wst:SignWith – /wst:RequestSecurityToken/wst:EncryptWith – /wst:RequestSecurityToken/wst:DelegateTo – /wst:RequestSecurityToken/wst:Forwardable – /wst:RequestSecurityToken/wst:Delegatable – /wst:RequestSecurityToken/wsp:Policy – /wst:RequestSecurityToken/wsp:PolicyReference
Response header	/wsa:Action Unsupported Responses: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Issue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Renew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Cancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Validate

Web Services Security specification - a chronology

The development of the Web Services Security specification includes information on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security specification. The OASIS Web Services Security specification serves as a basis for securing web services in WebSphere Application Server.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Advantages of using the JAX-WS programming model in WebSphere Application Server include:

- The configuration of qualities of service (QoS) is simplified when using policy sets. Policy sets combine configuration settings, including those for transport and message-level configuration. Policy sets and general bindings can be reused across multiple applications, making web services QoS more consumable.
- WS-Security for JAX-WS is supported in both a managed environment, such as a Java EE container, and unmanaged environments, such as Java Platform, Standard Edition (Java SE 6). In addition, there is an API for enabling WS-Security in the JAX-WS client.

Non-OASIS activities

Web services is gaining rapid acceptance as a viable technology for interoperability and integration. However, securing web services is one of the paramount quality of services that makes the adoption of web services a viable industry and commercial solution for businesses. IBM and Microsoft jointly published a security white paper on web services entitled Security in a Web Services World: A Proposed Architecture and Roadmap. The white paper discusses the following initial and subsequent specifications in the proposed Web Services Security roadmap:

Web service security
This specification defines how to attach a digital signature, use encryption, and use security tokens in SOAP messages.

WS-Policy
This specification defines the language that is used to describe security constraints and the policy of intermediaries or endpoints.

WS-Trust
This specification defines a framework for trust models to establish trust between web services.

WS-Privacy
This specification defines a model of how to express a privacy policy for a web service and a requester.

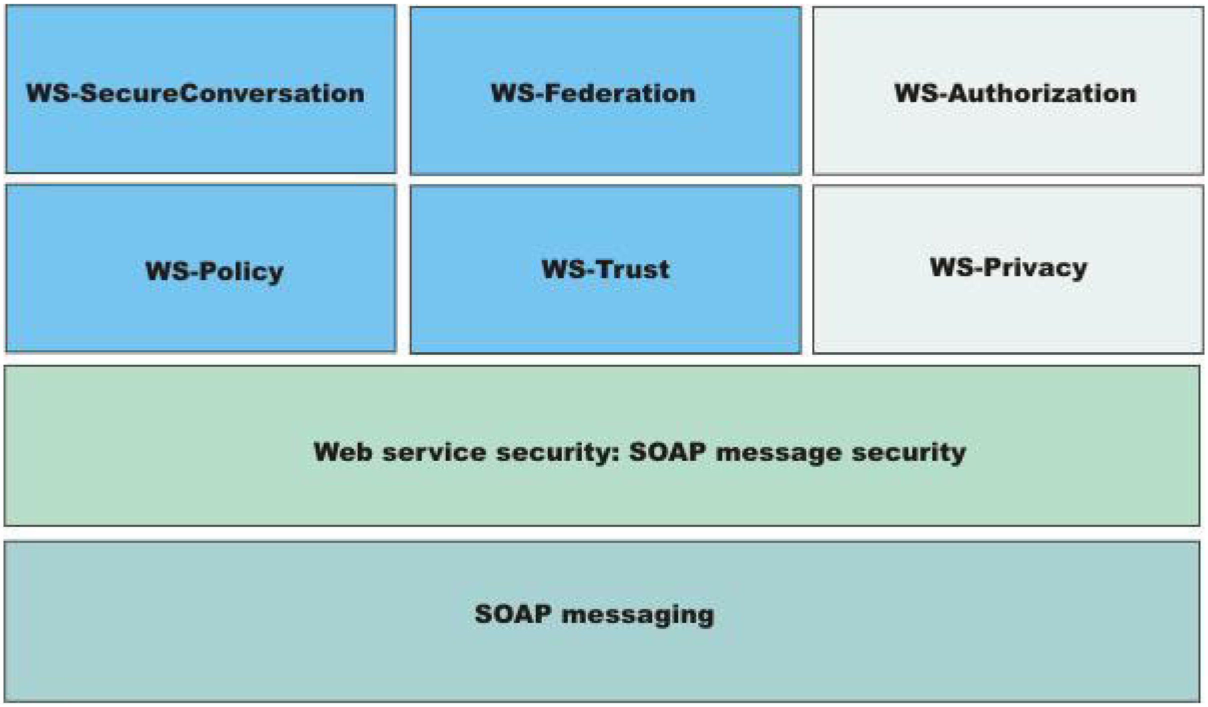
WS-SecureConversation
This specification defines how to exchange and establish a secured context, which derives session keys between web services.

WS-Authorization

This specification defines the authorization policy for a Web service. However, the WS-Authorization specification has not been published. The existing implementation of Web Services Security is based upon the Web Services for Java Platform, Enterprise Edition (Java EE) or Java Specification Requirements (JSR) 109 specification. The implementation of Web Services Security leverages the Java EE role-based authorization checks. For conceptual information, read about role-based authorization. If you develop a web service that requires method-level authorization checks, then you must use stateless session beans to implement your web service. For more information, read about securing enterprise bean applications.

If you develop a web service that is implemented as a servlet, you can use coarse-grained or URL-based authorization in the web container. However, in this situation, you cannot use the identity from Web Services Security for authorization checks. Instead, you can use the identity from the transport. If you use SOAP over HTTP, then the identity is in the HTTP transport.

This following figure shows the relationship between these specifications:



In April 2002, IBM, Microsoft, and VeriSign proposed the Web Services Security (WS-Security) specification on their websites as depicted by the green box in the previous figure. This specification included the basic ideas of a security token, XML digital signature, and XML encryption. The specification also defined the format for user name tokens and encoded binary security tokens. After some discussion and an interoperability test based on the specification, the following issues were noted:

- The specification requires that the Web Services Security processors understand the schema correctly so that the processor distinguishes between the ID attribute for XML digital signature and XML encryption.
- The freshness of the message, which indicates whether the message complies with predefined time constraints, cannot be determined.
- Digested password strings do not strengthen security.

In August 2002, IBM, Microsoft, and VeriSign published the *Web Services Security Addendum*, which attempted to address the previously listed issues. The following solutions were addressed in the addendum:

- Require a global ID attribute for XML signature and XML encryption.
- Use time stamp header elements that indicate the time of the creation, receipt, or expiration of the message.
- Use password strings that are digested with a time stamp and nonce, which is a randomly generated token.

The specifications for the blue boxes in the previous figure have been proposed by various industry vendors and various interoperability events have been organized by the vendors to verify and refine the proposed specifications.

OASIS activities

In June 2002, OASIS received a proposed Web Services Security specification from IBM, Microsoft, and VeriSign. The Web Services Security Technical Committee (WSS TC) was organized at OASIS soon after the submission. The technical committee included many companies including IBM, Microsoft, VeriSign, Sun Microsystems, and BEA Systems.

In September 2002, WSS TC published its first specification, Web Services Security Core Specification, Working Draft 01. This specification included the contents of both the original Web Services Security specification and its addendum.

The coverage of the technical committee became larger as the discussion proceeded. Because the Web Services Security Core Specification allows arbitrary types of security tokens, proposals were published as profiles. The profiles described the method for embedding tokens, including Security Assertion Markup Language (SAML) tokens and Kerberos tokens embedded into the Web Services Security messages. Subsequently, the definitions of the usage for user name tokens and X.509 binary security tokens, which were defined in the original Web Services Security Specification, were divided into the profiles.

WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1 support the following specifications:

- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)
- Web Services Security: Username Token Profile Draft 2

In April 2004, the Web Service Security specification (officially called Web Services Security: SOAP Message Security Version 1.0) became the Version 1.0 OASIS standard. Also, the Username token and X.509 token profiles are Version 1.0 specifications. WebSphere Application Server 6 and later support the following Web Services Security specifications from OASIS:

- Web Services Security: SOAP Message Security 1.0 specification
- Web Services Security: Username Token 1.0 Profile
- Web Services Security: X.509 Token 1.0 Profile

In February 2006, the core Web Service Security specification was updated and became the Version 1.1 OASIS standard. Also, the Username token, X.509 token profile, and Kerberos token profile were updated to the Version 1.1 specifications. Portions of the following Web Services Security specifications from OASIS are supported in WebSphere Application Server, specifically signature confirmation, encrypted header, and thumbprint references:

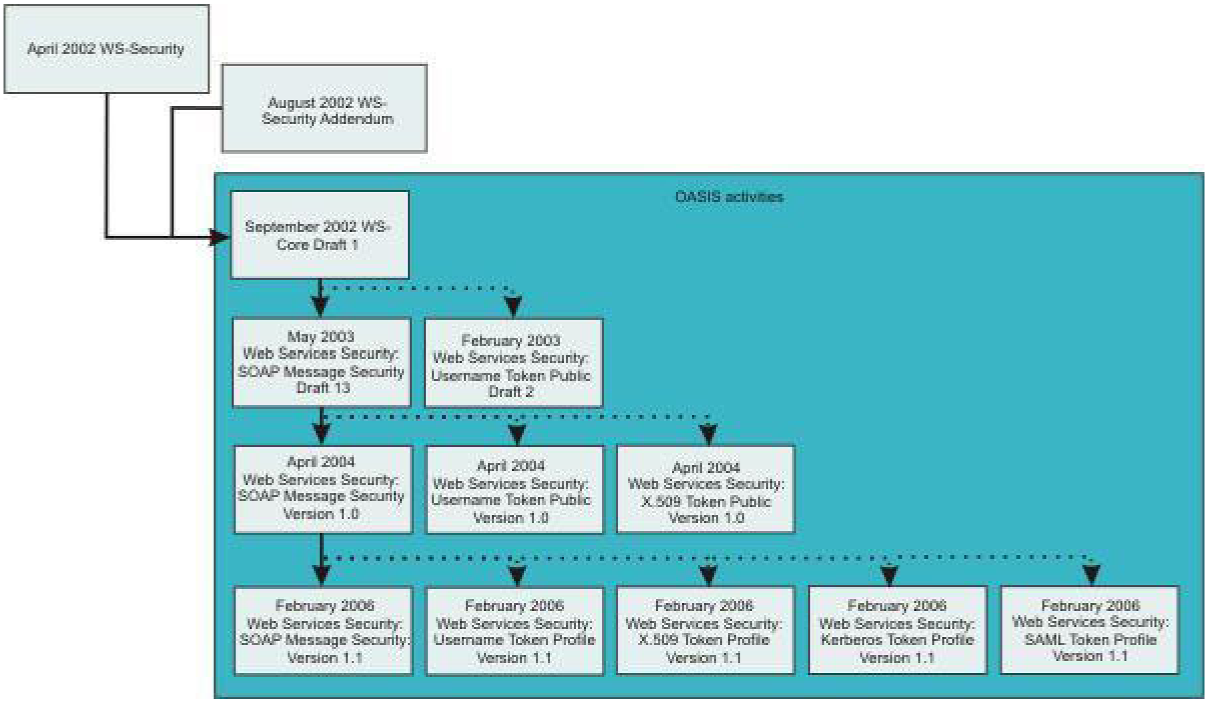
- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security UsernameToken Profile 1.1 OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security X.509 Certificate Token Profile 1.1 OASIS Standard Specification, 1 February 2006

The following specification describes the use of Kerberos tokens with respect to the Web Services Security message security specifications. The specification defines how to use a Kerberos token to support authentication and message protection: OASIS: Web Services Security Kerberos Token Profile 1.1 OASIS Standard Specification, 1 February 2006.

In 2007, the OASIS Web Services Secure Exchange Technical Committee (WS-SX) produced and approved the following specifications. Portions of these specifications are supported by WebSphere Application Server Version 7 and later.

- WS-SecureConversation
- WS-Trust
- WS-SecurityPolicy

The following figure shows the various Web Services Security-related specifications.


April 2002 WS-Security
August 2002 WS-Security Addendum
OASIS activities
September 2002 WS-Core Draft 1
May 2003 Web Services Security: SOAP Message Security Draft 13
February 2003 Web Services Security: Username Token Public Draft 2
April 2004 Web Services Security: SOAP Message Security Version 1.0
April 2004 Web Services Security: Username Token Public Version 1.0
April 2004 Web Services Security: X.509 Token Public Version 1.0
February 2006 Web Services Security: SOAP Message Security: Version 1.1
February 2006 Web Services Security: Username Token Profile Version 1.1
February 2006 Web Services Security: X.509 Token Profile Version 1.1
February 2006 Web Services Security: Kerberos Token Profile Version 1.1
February 2006 Web Services Security: SAML Token Profile Version 1.1

WebSphere Application Server also provides plug-in capability to enable security providers to extend the runtime capability and implement some of the higher level specifications in the Web Service Security stack. The plug-in points are exposed as Service Provider Programming Interfaces (SPI). For more information on these SPIs, see “Default implementations of the Web Services Security service provider programming interfaces” on page 936.

Web Services Security specification 1.0 development

The OASIS Web Services Security specification is based upon the following World Wide Web Consortium (W3C) specifications. Most of the W3C specifications are in the standard body recommended status.

- XML-Signature Syntax and Processing

  W3C recommendation, February 2002 (Also, IETF RFC 3275, March 2002)
- Canonical XML Version 1.0

  W3C recommendation, March 2001
- Exclusive XML Canonicalization Version 1.0

  W3C recommendation, July 2002
- XML-Signature XPath Filter Version 2.0

  W3C Recommendation, November 2002
- XML Encryption Syntax and Processing

  W3C Recommendation, December 2002
- Decryption Transform for XML Signature

  W3C Recommendation, December 2002

These specifications are supported in WebSphere Application Server in the context of Web Services Security. For example, you can sign a SOAP message by specifying the integrity option in the deployment descriptors. There is a client side application programming interface (API) that an application can use to enable Web Services Security for securing a SOAP message.

The OASIS Web Services Security Version 1.0 specification defines the enhancements that are used to provide message integrity and confidentiality. It also provides a general framework for associating the security tokens with a SOAP message. The specification is designed to be extensible to support multiple security token formats. The particular security token usage is addressed with the security token profile.

Specification and profile support in WebSphere Application Server

OASIS is working on various profiles. For more information, see Organization for the Advancement of Structured Information Standards Committees.

The following list includes of the published draft profiles and OASIS Web Services Security technical committee work in progress.

WebSphere Application Server does not support these profiles:

- Web Services Security: SAML token profile 1.0
- Web Services Security: Rights Expression Language (REL) token profile 1.0
- Web Services Security: SOAP Messages with Attachments (SwA) profile 1.0

Note: Support for Web Services Security draft 13 and Username token profile draft 2 is deprecated in WebSphere Application Server 5.0.2, 5.1.0 and 5.1.1. For migration information, see Migrating JAX-RPC Web Services Security applications to Version 8.5 applications.

The wire format of the SOAP message with Web Services Security in Web Services Security Version 1.0 has changed and is not compatible with previous drafts of the OASIS Web Services Security specification. Interoperability between OASIS Web Services Security Version 1.0 and previous Web Services Security drafts is not supported. However, it is possible to run an application that is based on Web Services Security draft 13 on WebSphere Application Server Version 6 and later. The application can interoperate with an application that is based on Web Services Security draft 13 on WebSphere Application Server Version 5.0.2, 5.1 or 5.1.1.

WebSphere Application Server supports both the OASIS Web Services Security draft 13 and the OASIS Web Services Security 1.0 specification. But in WebSphere Application Server Version 6 and later, the support of OASIS Web Services Security draft 13 is deprecated. However, applications that were developed using OASIS Web Services Security draft 13 on WebSphere Application Server 5.0.2, 5.1.0 and 5.1.1 can run on WebSphere Application Server Version 6 and later. OASIS Web Services Security Version 1.0 support is available only for Java Platform, Enterprise Edition (Java EE) Version 1.4 and later applications. The configuration format for the deployment descriptor and the binding is different from previous versions of WebSphere Application Server. You must migrate the existing applications to Java EE 1.4 and migrate the Web Services Security configuration to the WebSphere Application Server Version 6 format.

Other Web Services Security specifications development

The most recently updated versions of the following OASIS Web Services Security specifications are supported in WebSphere Application Server in the context of Web Services Security:

- WS-Trust Version 1.3

  The Web Services Trust Language (WS-Trust) uses the secure messaging mechanisms of Web Services Security to define additional primitives and extensions for the issuance, exchange and validation of security tokens. WS-Trust enables the issuance and dissemination of credentials within different trust domains. This specification defines ways to establish, assess the presence of, and broker trust relationships.
- WS-SecureConversation Version 1.3

  The Web Services Secure Conversation Language (WS-SecureConversation) is built on the WS-Security and WS-Policy models to provide secure communication between services. WS-Security focuses on the message authentication model but not a security context, and thus is subject several forms of security attacks. This specification defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation. By using the SOAP extensibility model, modular SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment.

- WS-SecurityPolicy Version 1.2

  Web Services Security Policy (WS-Policy) provides a general purpose model and syntax to describe and communicate the policies of a web service. WS-Policy assertions express the capabilities and constraints of a particular web service. WS-PolicyAttachments defines several methods for associating the WS-Policy expressions with web services (such as WSDL). The Web Services Security specifications have been updated following the re-publication of WS-Security Policy in July 2005, to reflect the constraints and capabilities of web services that are using WS-Security, WSTrust and WS-SecureConversation. WS-ReliableMessaging Policy has also been re-published in 2005 to express the capabilities and constraints of web services implementing WS-ReliableMessaging.

Web Services Interoperability Organization (WS-I) activities

Web Services Interoperability Organization (WS-I) is an open industry effort to promote web services interoperability across vendors, platforms, programming languages and applications. The organization is a consortium of companies across many industries including IBM, Microsoft, Oracle, Sun, Novell, VeriSign, and Daimler Chrysler. WS-I began working on the basic security profile (BSP) in the spring of 2003. BSP consists of a set of non-proprietary web services specifications that clarifies and amplifies those specifications to promote Web Services Security interoperability across different vendor implementations. As of June 2004, BSP is a public draft. For more information, see the Web Services Interoperability Organization web page.

Specifically, see Basic Security Profile Version 1.0 for details about the BSP. WebSphere Application Server supports compliance with the BSP draft, but Web Services Security does not support the BSP Version 1.1 draft. See “Basic Security Profile compliance tips” on page 968 for the details to configure your application in compliance with the BSP draft.

Web Services Security configuration considerations

To secure web services for WebSphere Application Server, you must specify several different configurations. Although there is not a specific sequence in which you must specify these different configurations, some configurations reference other configurations.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

You can configure Web Services Security on the application level and the server level. The following table shows an example of the relationships between each of the configurations that apply to just the application, to an entire server. However, the requirements for the bindings depend upon the deployment descriptor. Some binding information depends upon other information in the binding or server-level configuration. Within the table, the configurations in the Referenced configurations column are referenced by the configuration listed in the Configuration name column. For example, the token generator on the application-level for the request generator references the collection certificate store, the nonce, time stamp, and callback handler configurations.

Table 81. The relationship between the configurations.. Use the table to determine the mapping between the configurations and the level of Web Services Security.

Configuration level	Configuration name	Referenced configurations
Application-level request generator	Token generator	• Collection certificate store • Nonce • Timestamp • Callback handler
Application-level request generator	Key information	• Key locator • Key name • Token
Application-level request generator	Signing information	• Key information
Application-level request generator	Encryption information	• Key information
Application-level request consumer	Token consumer	• Trust anchor • Collection certificate store • Trusted ID evaluators • Java Authentication and Authorization Service (JAAS) configuration
Application-level request consumer	Key information	• Key locator • Token
Application-level request consumer	Signing information	• Key information
Application-level request consumer	Encryption information	• Key information
Application-level response generator	Token generator	• Collection certificate store • Callback handler
Application-level response generator	Key information	• Key locator • Token
Application-level response generator	Signing information	• Key information
Application-level response generator	Encryption information	• Key information
Application-level response consumer	Token consumer	• Trust anchor • Collection certificate store • JAAS configuration
Application-level response consumer	Key information	• Key locator • Key name • Token
Application-level response consumer	Signing information	• Key information
Application-level response consumer	Encryption information	• Key information
Server-level default generator bindings	Token generator	• Collection certificate store • Callback handler
Server-level default generator bindings	Key information	• Key locator • Token
Server-level default generator bindings	Signing information	• Key information
Server-level default generator bindings	Encryption information	• Key information
Server-level default consumer bindings	Token consumer	• Trust anchor • Collection certificate store • Trusted ID evaluator • JAAS configuration
Server-level default consumer bindings	Key information	• Key locator • Token
Server-level default consumer bindings	Signing information	• Key information

Table 81. The relationship between the configurations. (continued). Use the table to determine the mapping between the configurations and the level of Web Services Security.

Configuration level	Configuration name	Referenced configurations
Server-level default consumer bindings	Encryption information	• Key information

When multiple applications will use the same binding information, consider configuring the binding information on the server level. For example, you might have a global key locator configuration that is used by multiple applications. Configuration information for the application-level precedes similar configuration information on the server-level.

Default bindings and runtime properties for Web Services Security

Use this page to configure the settings for nonce on the server level and to manage the default bindings for the signing information, encryption information, key information, token generators, token consumers, key locators, collection certificate store, trust anchors, trusted ID evaluators, algorithm mappings, and login mappings.

Displayed options and the panel title depend on your server configuration and version.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using WebSphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web Services Security**.

Read the web services documentation before you begin defining the default bindings for Web Services Security.

Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens.

In WebSphere Application Server and WebSphere Application Server, Express, you must specify values for the Nonce cache timeout, Nonce maximum age, and Nonce clock skew fields for the server level.

Nonce cache timeout

Specifies the timeout value, in seconds, for the nonce cached on the server. Nonce is a randomly generated value.

The Nonce cache timeout field is required.

If you make changes to the value for the Nonce cache timeout field, you must restart the application server for the changes to take effect.

Information	Value
Default	600 seconds
Minimum	300 seconds

Nonce maximum age

Specifies the default time, in seconds, before the nonce timestamp expires. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce cache timeout field for the server level.

The Nonce maximum age field is required.

Information	Value
Default	300 seconds
Range	300 to the value that is specified, in seconds, in the Nonce cache timeout field.

Nonce clock skew

Specifies the default clock skew value, in seconds, to consider when the application server checks the timeliness of the message. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce maximum age field.

The Nonce clock skew field is required.

Information	Value
Default	0 seconds
Range	0 to the value that is specified, in seconds, in the Nonce maximum age field.

Enable cryptographic operations on hardware device

Enables cryptographic operations on hardware devices. Enabling this feature might improve the performance, depending on the hardware device.

Cryptographic hardware configuration name

Specifies the name of the hardware device configuration name that is defined in the keystore settings in the secure communications.

This value is necessary only if **Hardware acceleration** has been selected.

Custom properties

The linked Properties panel specifies additional properties for the security runtime configuration.

Web Services Security provides message integrity, confidentiality, and authentication

OASIS Web Services Security (WS-Security) is a flexible standard that is used to secure web services at the message level within multiple security models. You can secure SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens.

The WS-Security specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Message-level security, or securing web services at the message level, addresses the same security requirements as for traditional web security. These security requirements include: identity, authentication, authorization, integrity, confidentiality, nonrepudiation, basic message exchange, and so forth. Both traditional web and message-level security share many of the same mechanisms for handling security, including digital certificates, encryption, and digital signatures. While HTTPS and Secure Sockets Layer (SSL) transport-level technology may be used for securing web services, some security scenarios are addressed more effectively by message-level security.

Traditional web security mechanisms, such as HTTPS, might be insufficient to manage the security requirements of all web service scenarios. For example, when an application sends a document with JAX-RPC using HTTPS, the message is secured only for the HTTPS connection, meaning during the transport of the document between the service requester (the client) and the service. However, the application might require that the document data be secured beyond the HTTPS connection, or even beyond the transport layer. By securing web services at the message level, message-level security is capable of meeting these expanded requirements.

Message-level security applies to XML documents that are sent as SOAP messages. Message-level security makes security part of the message itself by embedding all required security information in the SOAP header of a message. In addition, message-level security can apply security mechanisms, such as encryption and digital signature, to the data in the message itself.

With message-level security, the SOAP message itself either contains the information needed to secure the message or it contains information about where to get that information to handle security needs. The SOAP message also contains information relevant to the protocols and procedures for processing the specified message-level security. However, message-level security is not tied to any particular transport mechanism. Because the security information is part of the message, it is independent of a transport protocol, such as HTTPS.

The client adds to the SOAP message header security information that applies to that particular message. When the message is received, the web service endpoint, using the security information in the header, verifies the secured message and validates it against the policy. For example, the service endpoint might verify the message signature and check that the message has not been tampered with. It is possible to add signature and encryption information to the SOAP message headers, as well as other information such as security tokens for identity (for example, an X.509 certificate) that are bound to the SOAP message content.

For WebSphere Application Server Versions 6 and later, Web Services Security can be applied as transport-level security and as message-level security. You can architect highly secure client and server designs by using these security mechanisms. Transport-level security refers to securing the connection between a client application and a web service with Secure Sockets Layer (SSL).

You can apply various scenarios of Web Services Security according to the characteristics of each web service application. You have choices of how to protect your information when using Web Services Security. The authentication mechanism, integrity, and confidentiality can be applied at the message level and at the transport level. When message-level security is applied, you can protect the SOAP message with a security token, digital signature, and encryption.

Without Web Services Security, the SOAP message is sent in clear text, and personal information such as a user ID or an account number is not protected. Without applying Web Services Security, there is only a SOAP body under the SOAP envelope in the SOAP message. By applying features from the WS-Security specification, the SOAP security header is inserted under the SOAP envelope in the SOAP message when the SOAP body is signed and encrypted.

To maintain the integrity or confidentiality of the message, digital signatures and encryption are typically applied.

- *Confidentiality* specifies the confidentiality constraints that are applied to generated messages. This includes specifying which message parts within the generated message must be encrypted, and the message parts to attach encrypted Nonce and time stamp elements to.
- *Integrity* is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption. Multiple signatures and encryptions are supported. In addition, both signing and encryption can be applied to the same parts, such as the SOAP body.

You can add an authentication mechanism by inserting various types of security tokens, such as the Username token (<UsernameToken> element). When the Username token is received by the web service server, the user name and password are extracted and verified. Only when the user name and password combination is valid, will the message be accepted and processed at the server. Using the Username token is just one of the ways of implementing authentication. This mechanism is also known as *basic authentication*.

In addition to digital signatures, encryption, and basic authentication, other forms of authentication include identity assertion, LTPA tokens, Kerberos tokens, and custom tokens. These other forms of authentication are also extensions of WebSphere Application Server. You can configure these authentication mechanisms using the assembly tools to implement authentication.

With updates to Web Services Security in the Version 1.1 specification, it is possible to layer additional functionality in addition to these basic mechanisms. Some Version 1.1 mechanisms are extensions of WebSphere Application Server, such as signature confirmation and the encrypted header. The security token profiles that are supported by WebSphere Application Server include the Username token profile, the X.509 token profile, and the Kerberos profile. In this case, when the message is received, the web service endpoint, using the security information in the header, applies the appropriate security mechanisms to the message. For example, the service endpoint might add signature and encryption information to the SOAP message headers, as well as other information, such as security tokens, that are bound to the SOAP message content. You can implement these new mechanisms by using a policy set.

WS-SecureConversation was introduced in WebSphere Application Server Version 6.1 with the Feature Pack for Web Services. Secure Conversation uses a session key to protect SOAP messages more efficiently, particularly when multiple SOAP messages are transmitted in a session.

Other enhancements include:

- The Kerberos token, which is used for both authentication and for subsequent message protection.
- Dynamic policy, which allows the client to retrieve the provider policy through a WSDL request, or using Web Services MetadataExhange (WS-MEX), to simplify web services client deployment.

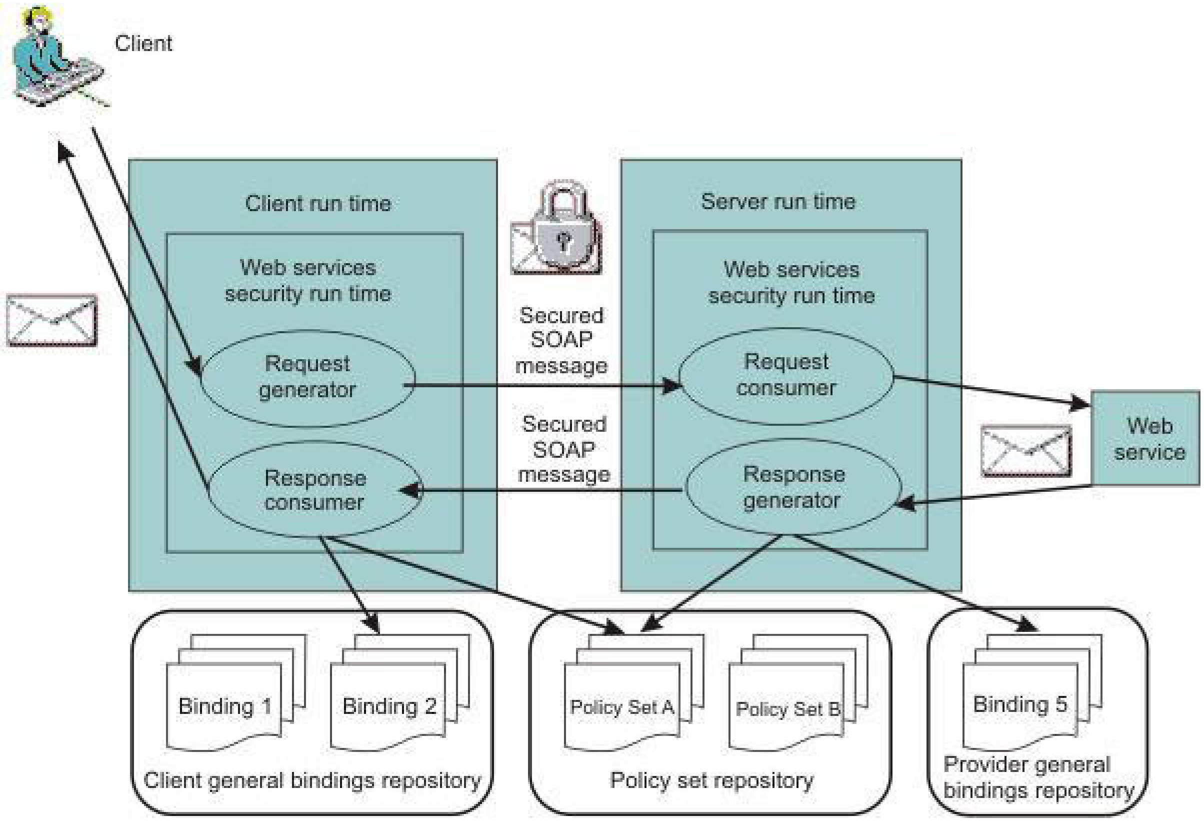
High-level architecture for Web Services Security

The Web Services Security policy is specified in the IBM extension of the web services deployment descriptors when using the JAX-RPC programming model, and in policy sets when using the JAX-WS programming model. A stand-alone JAX-WS client application may specify Web Services Security policy programmatically. Binding data that supports the Web Services Security policy are stored in the IBM extension of the web services deployment descriptors for both the JAX-RPC and JAX-WS programming models. The Web Services Security run time enforces the security assertions that are specified in the policy document, or in the application program, in that order.

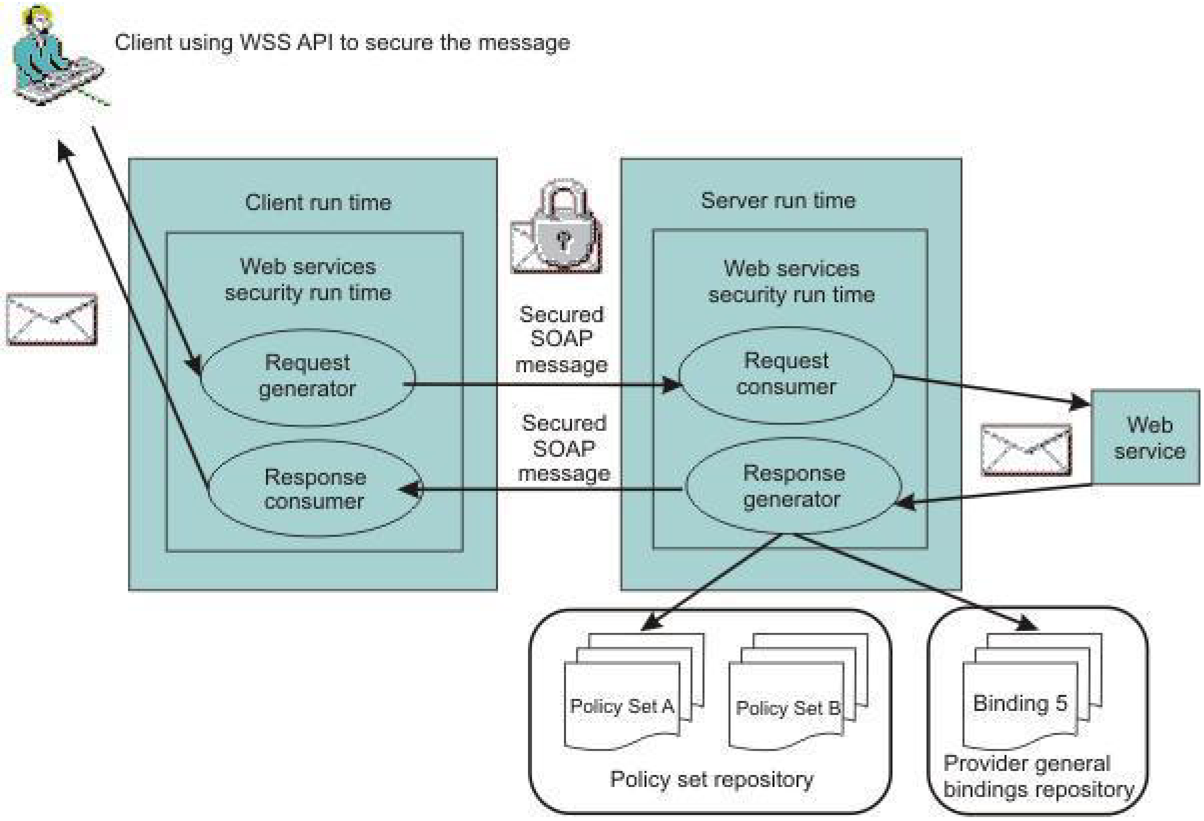
best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

WebSphere Application Server uses the Java Platform, Enterprise Edition (Java EE) Version 1.4 or later web services deployment model to implement Web Services Security. One of the advantages of deployment model is that you can define the Web Services Security requirements outside of the application business logic. With the separation of roles, the application developer can focus on the business logic and the security expert can specify the security requirement.

The following figure shows the high-level architecture model that is used to secure web services in WebSphere Application Server:



The WSS API can also be used to secure the message, as illustrated later in this section:



There are two sets of configurations on both the client side and the server side:

Request generator
This client-side configuration defines the Web Services Security requirements for the outgoing SOAP message request. These requirements might involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request generator was known as the *request sender*.

Request consumer
This server-side configuration defines the Web Services Security requirements for the incoming SOAP message request. These requirements might involve verifying that the required integrity parts are digitally signed; verifying the digital signature; verifying that the required confidential parts were encrypted by the request generator; decrypting the required confidential parts; validating the security tokens, and verifying that the security context is set up with the appropriate identity. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request consumer was known as the *request receiver*.

Response generator
This server-side configuration defines the Web Services Security requirements for the outgoing SOAP message response. These requirements might involve generating the SOAP message response with Web Services Security; including digital signature; and encrypting and attaching the security tokens, if necessary. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response generator was known as the *response sender*.

Response consumer
This client-side configuration defines the Web Services Security requirements for the incoming SOAP response. The requirements might involve verifying that the integrity parts are signed and the signature is verified; verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response consumer was known as the *response receiver*.

WebSphere Application Server does not include security policy negotiation or exchange between the client and server. This security policy negotiation, as defined by the WS-Policy, WS-PolicyAssertion, and WS-SecurityPolicy specifications, are not supported in WebSphere Application Server.

Note: The Web Services Security requirements that are defined in the request generator must match the request consumer. The requirements that are defined in the response generator must match the response consumer. Otherwise, the request or response is rejected because the Web Services Security constraints cannot be met by the request consumer and response consumer.

The format of the Web Services Security deployment descriptors and bindings are IBM proprietary. However, the following tools are available to edit the deployment descriptors and bindings:

IBM assembly tools
Use IBM assembly tools to edit the Web Services Security deployment descriptor and binding. Use the tools to assemble both web and Enterprise JavaBeans (EJB) modules. For more information, read about assembly tools.

WebSphere Application Server Administrative Console
Use this tool to edit the Web Services Security binding of a deployed application.

Security model mixture:

There can be multiple protocols and channels in the WebSphere Application Server Version 6 and later programming environments. Each of these applications serve different business needs.

For example, you might access:

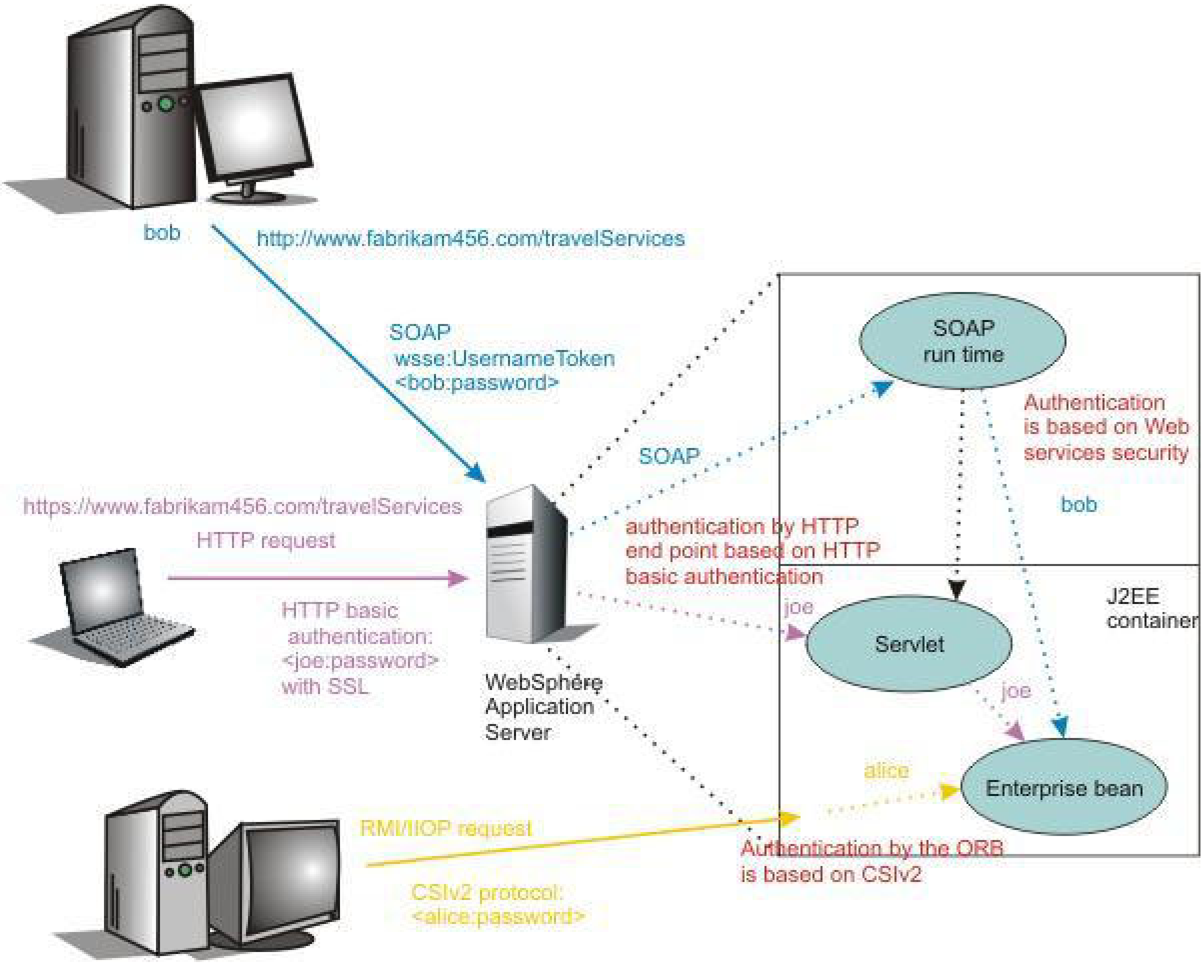
- A Web-based application through the HTTP transport such as a servlet, JavaServer Pages (JSP) file, HTML and so on.
- An enterprise application through the Remote Method Invocation (RMI) over the Internet Inter-ORB (RMI/IIOP) protocol.
- A web service application through the SOAP over HTTP, SOAP over the Java Message Service (JMS), or SOAP over the RMI/IIOP protocol.

More importantly, web services are often implemented as servlets with a Enterprise JavaBeans (EJB) file. Therefore, you can mix and match the Web Services Security model with the Java Platform, Enterprise Edition (Java EE) security model for web and EJB components. It is intended that web service security complement the Java EE role-based security and the security run time for WebSphere Application Server Version 6 and later.

Web Services Security also can take advantage of the security features in Java EE and the security run time for WebSphere Application Server Version 6 and later. For example, Web Services Security can use the following security features to provide an end-to-end security deployment:

- Use the local OS, Lightweight Directory Access Protocol (LDAP), and custom user registries for authenticating the username token
- Propagate the Lightweight Third Party Authentication (LTPA) security token in the SOAP message
- Use identity assertion
- Use a trust association interceptor (TAI)
- Enable security attribute propagation
- Use Java EE role-based authorization
- Use a Java Authorization Contract for Containers (JACC) authorization provider, such as Tivoli Access Manager

The following figure shows that different security protocols are used to send authentication information to the application server. For a web service, you might use either HTTP basic authentication with Secure Sockets Layer (SSL) or a Web Services Security username token with signing and encryption. In the following figure, when identity *bob* from Web Services Security is authenticated and set as the caller identity of the SOAP message request, the Java EE Enterprise JavaBeans container performs authorization using *bob* before the call is dispatched to the service implementation, which, in this case, is the enterprise bean.



You can secure a web service using the transport layer security. For example, when you are using SOAP over HTTP, HTTPS can be used to secure the web service. However, transport layer security provides point-to-point security only. This layer of security might be adequate for certain scenarios. However, when the SOAP message must travel through intermediary servers (multi-hop) before it is consumed by the target endpoint, you might use SOAP over the Java Message Service (JMS). The usage scenarios and security requirements dictate how to secure web services. The requirements depend upon the operating environment and the business needs. However, one key advantage of using Web Services Security is that it is transport layer independent; the same Web Services Security constraints can be used for SOAP over HTTP, SOAP over JMS, or SOAP over RMI/IIOP.

Overview of platform configuration and bindings:

The Web Services Security policy is specified in the IBM extension of the web services deployment descriptors when using the JAX-RPC programming model, and in policy sets when using the JAX-WS programming model. Binding information to support the Web Services Security policy is stored in the IBM extension of the web services deployment descriptors for both the JAX-RPC and JAX-WS programming models.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the

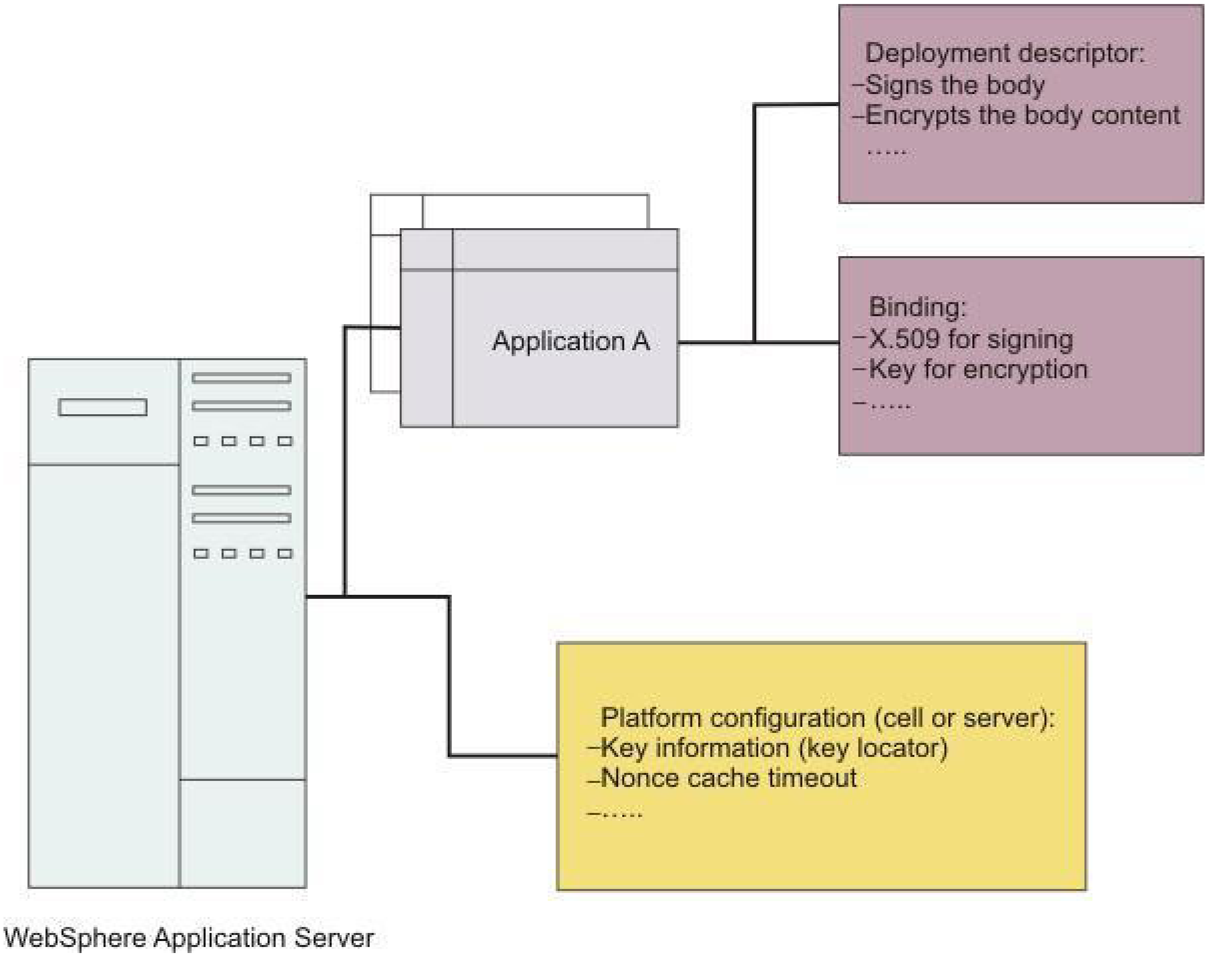
JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Due to the complexity of these files, it is not recommended that you edit the deployment descriptor and binding files manually with a text editor because they might cause errors. It is recommended, however, that you use the tools provided by IBM to configure the Web Services Security constraints for an application. These tools are the WebSphere Application Server administrative console, or an assembly tool. For more information about IBM assembly tools, see the assembly tools information.

You can use the policy set function of the WebSphere Application Server to simplify your web services configuration because policy sets group security and other web services settings into reusable units. Policy sets are assertions about how quality of services is defined. A policy set incorporates policy types, and their settings.

In addition to the application deployment descriptor and binding files, WebSphere Application Server Versions 6 and later have a server level Web Security Services configuration. These configurations are global for all applications. Because WebSphere Application Server Version 6 and later support 5.x applications, some of the configurations are valid for Version 5.x applications only and some are valid for Version 6 and later applications only.

The following figure represents the relationship of the application deployment descriptor and binding files to the cell (WebSphere Application Server, Network Deployment only) or server level configuration.



Platform configuration

The following options are available in the administrative console:

Nonce cache timeout
This option, which is found on the cell level (WebSphere Application Server, Network Deployment only) and server level, specifies the cache timeout value for a nonce in seconds.

Nonce maximum age
This option, which is found on the cell level (WebSphere Application Server, Network Deployment only) and server level, specifies the default life span for the nonce in seconds.

Nonce clock skew
This option, which is found on the cell level (WebSphere Application Server, Network Deployment only) and server level, specifies the default clock skew to account for network delay, processing delay, and so on. It is used to calculate when the nonce expires. Its unit of measurement is seconds.

Distribute nonce caching
This feature enables you to distribute the cache for the nonce to different servers in a cluster. It is available for WebSphere Application Server Version 6.0.x and later.

The following features can be referenced in the application binding:

Key locator
This feature specifies how the keys are retrieved for signing, encryption, and decryption. The implementation classes for the key locator are different in WebSphere Application Server Versions 6 and later and Version 5.*x*.

Collection certificate store
This feature specifies the certificate store for certificate path validation. It is typically used for validating X.509 tokens during signature verification or constructing the X.509 token with a certificate revocation list that is encoded in the PKCS#7 format. The certificate revocation list is supported for WebSphere Application Server Version 6.x and later applications only.

Trust anchors
This feature specifies the trust level for the signer certificate and is typically used in the X.509 token validation during signature verification.

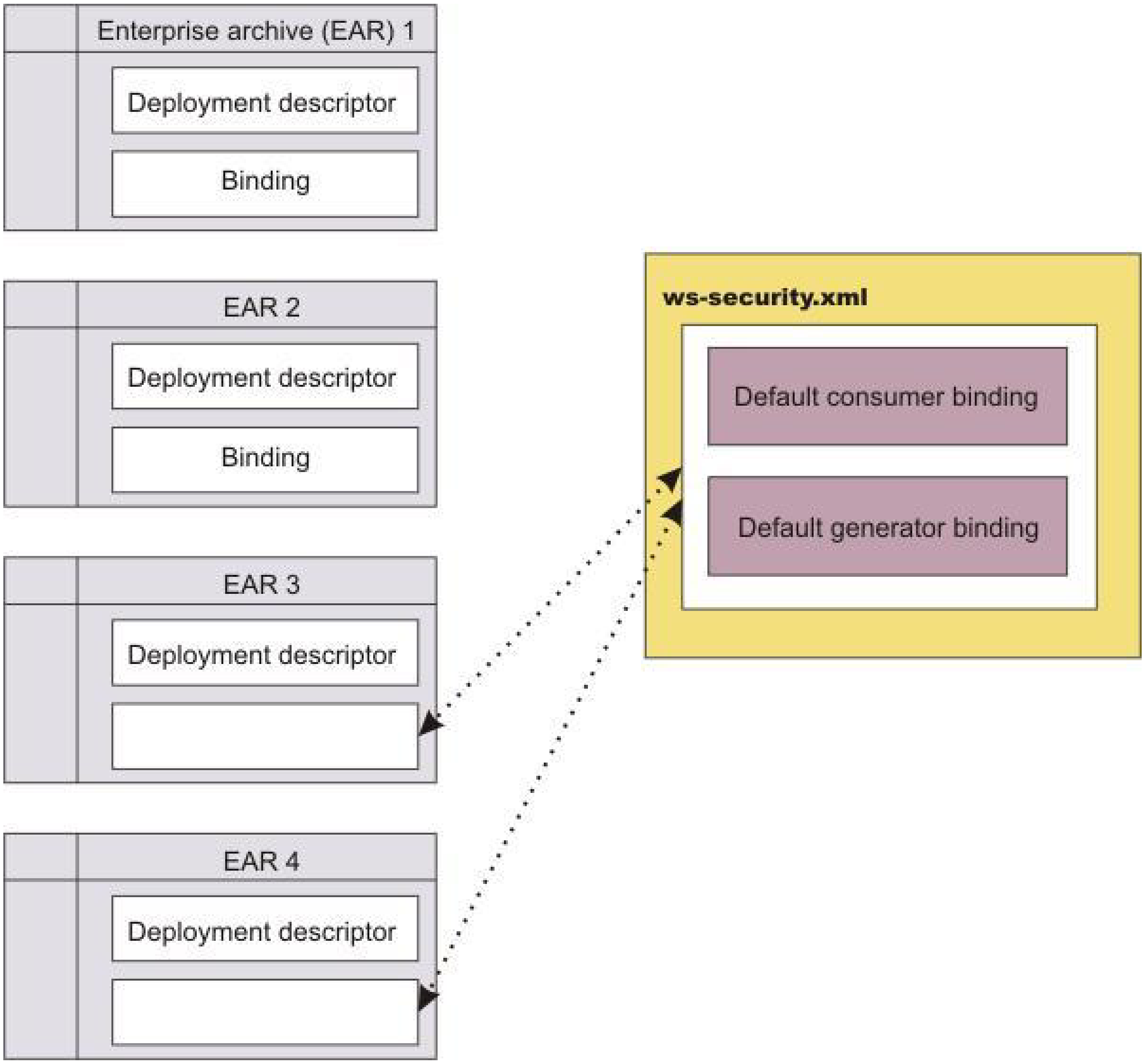
Trusted ID evaluators
This feature specifies how to verify the trust level for the identity. The feature is used with identity assertion.

Login mappings
This feature specifies the login configuration binding to the authentication methods. This feature is used by WebSphere Application Server Version 5.*x* applications only and it is deprecated.

Default bindings

The configuration of the default cell level and default server level bindings has changed in WebSphere Application Server. Previously, you could configure only one set of default bindings for the cell, and optionally configure one set of default bindings for each server. In version 7.0 and later, you can configure one or more general provider bindings and one or more general client bindings. However, only one general provider binding and one general client binding can be designated as the default.

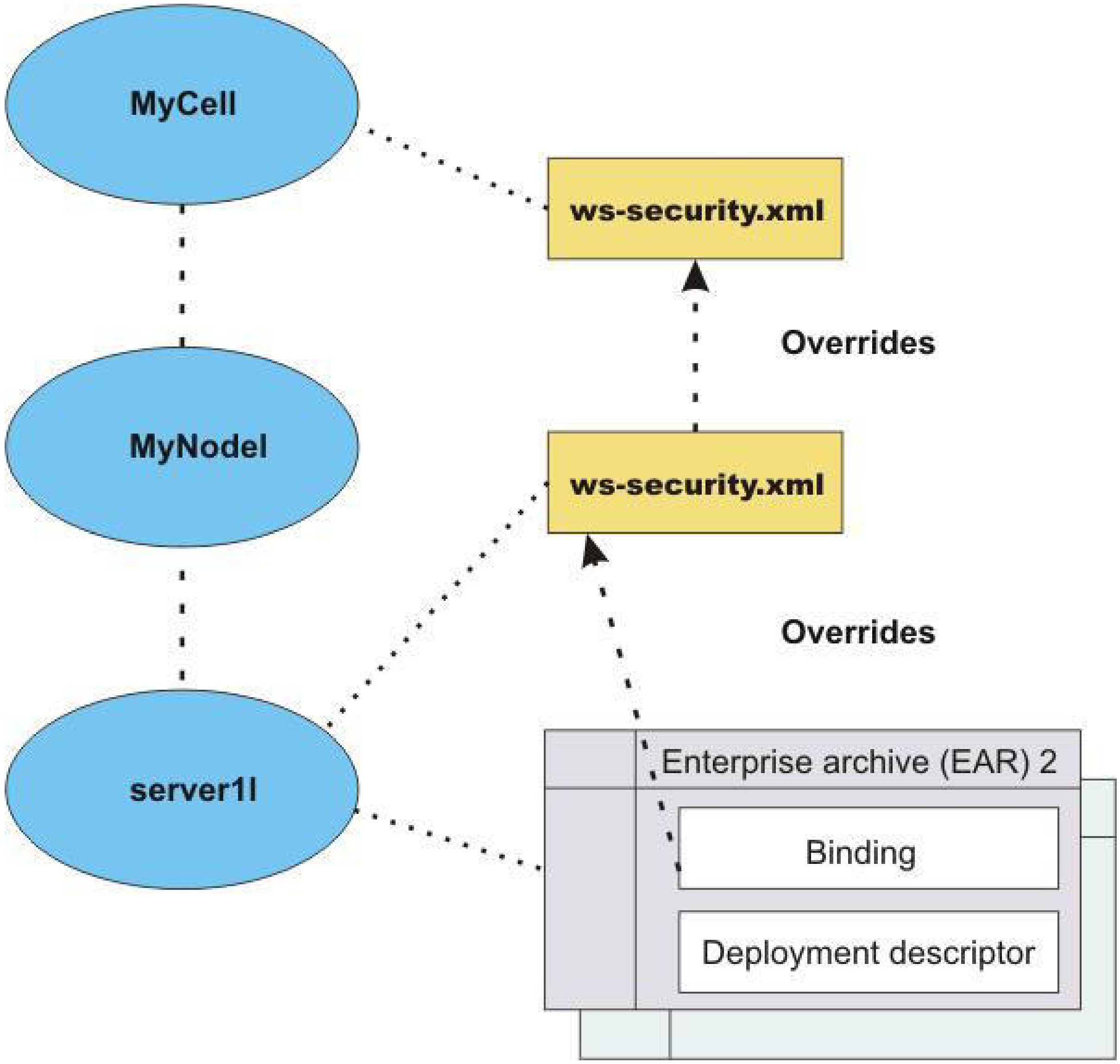
The following figure shows the relationship between the application enterprise archive (EAR) file and the `ws-security.xml` file.



Applications EAR 1 and EAR 2 have specific bindings in the application binding file. However, applications EAR 3 and EAR 4 do not have a binding in the application binding file; it must be referenced to use the default bindings defined in the `ws-security.xml` file. The configuration is resolved by nearest configuration in the hierarchy. For example, there might be key locators named `mykeylocator` that is defined in the application binding file and the server level.

If `mykeylocator` is referenced in the application binding, then the key locator that is defined in the application binding is used. The visibility scope of the data depends upon where the data is defined. If the data is defined in the application binding, then its visibility is scoped to that particular application. If the data is defined on the server level, then the visibility scope is all of the applications deployed on that server. In general, if data is not meant to be shared by other applications, define the configuration in the application binding level.

The following figure shows the relationship of the bindings on the application, server, and cell (WebSphere Application Server, Network Deployment only) levels.



General bindings

General bindings are used as the default bindings at the cell level or server level. The general bindings that are shipped with WebSphere Application Server are initially set as the default bindings, but you can choose a different binding as the default, or change the level of binding that should be used as the default, for example, from cell level binding to server level binding.

In version 7.0 and later, there are two types of bindings: application specific bindings, and general bindings. Both types of bindings are supported for WS-Security policy sets. General bindings can be shared across multiple applications and for trust service attachments. There are two types of general bindings: one for service providers and one for service clients. Multiple general bindings can be defined for the provider and also for the client.

Keys:

Use keys for XML digital signature and encryption.

There are two predominant kinds of keys used in the current Web Services Security implementation:

- Public key - such as Rivest Shamir Adleman (RSA) encryption and Digital Signature Algorithm (DSA) encryption
- Secret key - such as triple-strength DES (3DES) encryption

In public key-based signature, a message is signed using the sender private key and is verified using the sender public key. In public key-based encryption, a message is encrypted using the receiver public key and is decrypted using the receiver private key. In secret key-based signature and encryption, the same key is used by both parties.

While the current implementation of Web Services Security can support both kinds of keys, the format of the message differs slightly between public key-based encryption and secret key-based encryption.

Key locator:

A key locator is an abstraction of the mechanism that retrieves the key for digital signature and encryption. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side.

Retrieve keys from one of the following sources, depending upon your implementation:
- Java keystore file
- Database
- Kerberos KDC server (WebSphere Application Server using JAX-WS only)
- Trust service can provide a security context token and key (WebSphere Application Server using JAX-WS only)

Key locators search for the key using some type of a clue. The following types of clues are supported:
- A string label of the key, which is explicitly passed through the application programming interface (API). The relationship between each key and its name (string label) is maintained inside the key locator.
- The implementation context of the key locator; explicit information is not passed to the key locator. A key locator determines the appropriate key according to the implementation context.

WebSphere Application Server Versions 6 and later support a secret key-based signature called HMAC-SHA1. If you use HMAC-SHA1, the SOAP message does not contain a binary security token. In this case, it is assumed that the key information within the message contains the key name that is used to specify the secret key within the keystore.

Because the key locators support the public key-based signature, the key for verification is embedded in the X.509 certificate as a `<BinarySecurityToken>` element in the incoming message. For example, key locators can obtain the identity of the caller from the context and can retrieve the public key of the caller for response encryption.

This section describes the usage scenarios for key locators.

Signing:

The name of the signing key is specified in the Web Services Security configuration. This value is passed to the key locator and the actual key is returned. The corresponding X.509 certificate also can be returned.

Verification:

By default, WebSphere Application Server Versions 6 and later supports the following types of key locators:

KeyStoreKeyLocator
Uses the keystore to retrieve the key that is used for digital signature and verification or encryption and decryption.

X509CertKeyLocator
Uses an X.509 certificate within a message to retrieve the key for verification or decryption.

SignerCertKeyLocator
Uses the X.509 certificate within the request message to retrieve the key that is used for encryption in the response message.

Encryption:

The name of the encryption key is specified in the Web Services Security configuration. This value is passed to the key locator and the actual key is returned. On the server side, you can use the SignerCertKeyLocator to retrieve the key for encryption in the response message from the X.509 certificate in the request message.

Decryption:

The Web Services Security specification recommends using the key identifier instead of the key name. However, while the algorithm for computing the identifier for the public keys is defined in Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280, there is no agreed-upon algorithm for the secret keys. Therefore, the current implementation of Web Services Security uses the identifier only when public key-based encryption is performed. Otherwise, the ordinal key name is used.

When you use public key-based encryption, the value of the key identifier is embedded in the incoming encrypted message. Then, the Web Services Security implementation searches for all of the keys managed by the key locator and decrypts the message using the key whose identifier value matches the one in the message.

When you use secret key-based encryption, the value of the key name is embedded in the incoming encrypted message. The Web Services Security implementation asks the key locator for the key with the name that matches the name in the message and decrypts the message using the key.

Trust anchor:

A trust anchor specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the SOAP message.

When using WebSphere Application Server with the JAX-RPC programming model, key stores are implemented with the following message points to validate the X.509 certificate that is used for digital signature or XML encryption:

- Request consumer, as defined in the `ibm-webservices-bnd.xmi` file.
- Response consumer, as defined in the `ibm-webservicesclient-bnd.xmi` file when a web service is acting as a client to another web service.

For WebSphere Application Server Version 7.0 and later, using JAX-WS, key stores are used by the following message points to validate the X.509 certificate that is used for digital signature or XML encryption:

- Request consumer, as defined in the inbound keys and certificates of the WS-Security bindings.
- Response consumer, as defined in the inbound keys and certificates of the WS-Security bindings when a web service is acting as a client to another web service.

Key stores are critical to the integrity of the digital signature validation. If the key stores are tampered with, the result of the digital signature verification is doubtful and compromised. Therefore, it is recommended that you secure the key stores. The binding configuration specified for the consumer must match the binding configuration for the generator.

The trust anchor is defined as java.security.cert.TrustAnchor in the Java CertPath application programming interface (API). The Java CertPath API uses the trust anchor and the certificate store to validate the incoming X.509 certificate that is embedded in the SOAP message. The Web Services Security implementation in WebSphere Application Server supports this trust anchor. In WebSphere Application Server, the trust anchor is represented as a Java key store object. The type, path, and password of the key store are passed to the implementation through the administrative console or by scripting.

Trusted ID evaluator:

A trusted ID evaluator is the mechanism that evaluates whether a given ID name is trusted.

Using the trusted ID evaluator with the JAX-RPC programming model

In the JAX-RPC programming model, the trusted ID evaluator, com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, is an abstraction of the mechanism that evaluates whether a given ID name is trusted. There are two trust modes for validating the trust of the upstream server when using JAX-RPC:

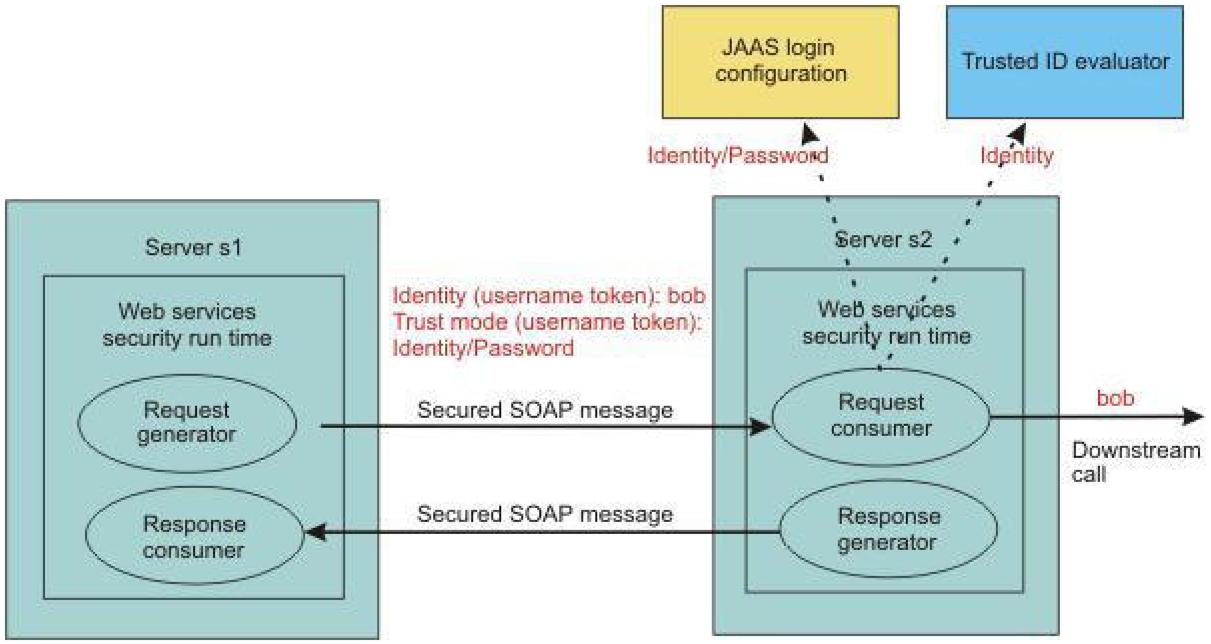
Basic authentication (username token)

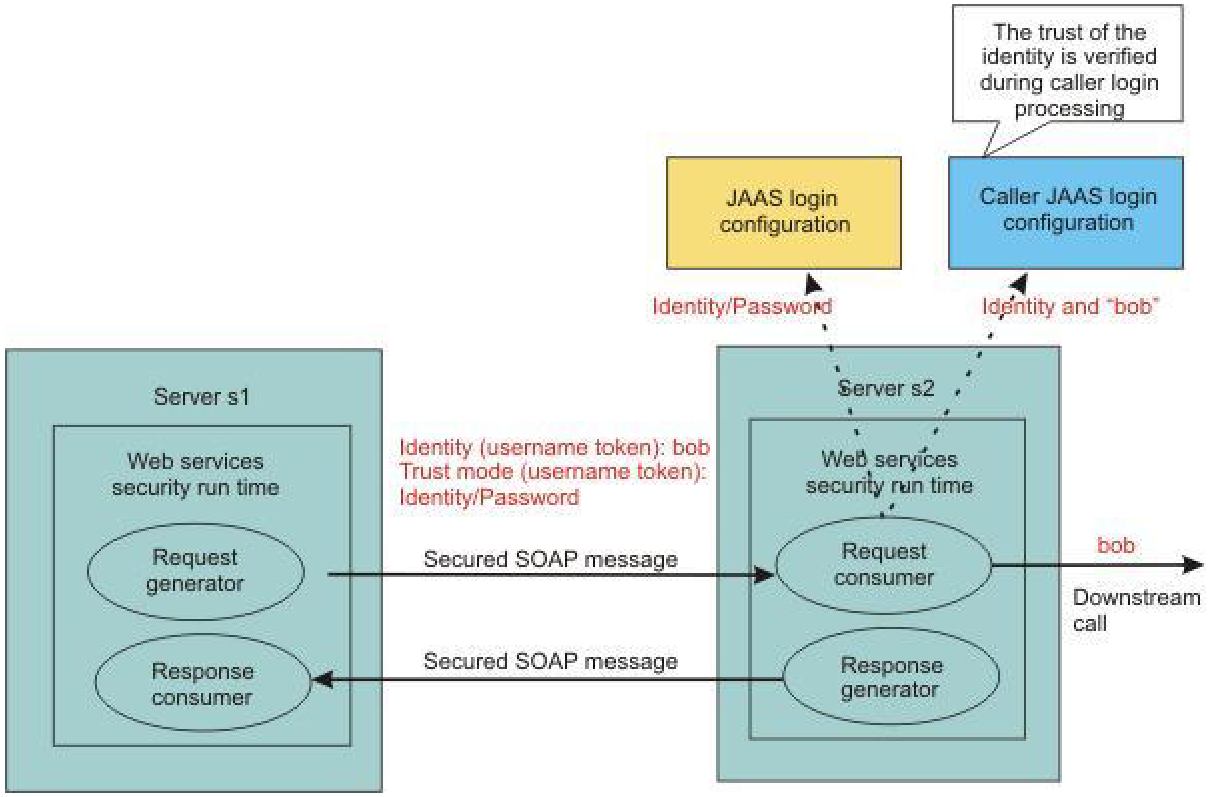
The upstream server sends a username token with a user name and password to a downstream server. The consumer or receiver of the message authenticates the username token and validates the trust based upon the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface.

Signature

The upstream server signs the message, which can be any message part such as the SOAP body. The upstream server sends the X.509 token to a downstream server. The consumer or receiver of the message verifies the signature and validates the X.509 token. The identity or the distinguished name from the X.509 token that is used in the digital signature is validated based on the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface. For the X.509 certificate, WebSphere Application Server uses the distinguished name in the certificate as a requester identity.

The following figures demonstrate the identity assertion trust process for both programming models:




The trust of the identity is verified during caller login processing
JAAS login configuration
Caller JAAS login configuration
Identity/Password
Identity and "bob"
Server s1
Web services security run time
Request generator
Response consumer
Identity (username token): bob
Trust mode (username token): Identity/Password
Secured SOAP message
Secured SOAP message
Server s2
Web services security run time
Request consumer
Response generator
bob
Downstream call

In this figure, server `s1` is the upstream server and identity assertion is set up between server `s1` and server `s2`. The `s1` server authenticates the identity called *bob*. Server `s1` wants to send *bob* to the `s2` server with a password. The trust mode is an `s1` credential that contains the identity and a password. Server `s2` receives the request, authenticates the user using a Java Authentication and Authorization Service (JAAS) login module, and uses the trusted ID evaluator to determine whether to trust the identity. If the identity is trusted, *bob* is used as the caller that invokes the service. If authorization is required, *bob* is the identity that is used for authorization verification.

The identity can be asserted as the RunAs (invocation) identity of the current security context. For example, the web services gateway authenticates a requester using a secure method such as password authentication and then sends the requester identity only to a back-end server. You might also use identity assertion for interoperability with another Web Services Security implementation.

Depending upon the implementation of JAX-RPC, you can use various types of infrastructure to store a list of the trusted IDs, such as:

- Plain text file
- Database
- Lightweight Directory Access Protocol (LDAP) server

The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment. The Web Services Security implementation invokes the trusted ID evaluator and passes the identity name of the intermediary as a parameter. If the identity is evaluated and deemed trustworthy, the procedure continues. Otherwise, an exception is created and the procedure is stopped.

Using the trusted ID evaluator with the JAX-WS programming model

In the JAX-WS programming model, the same concepts are supported for the trusted ID evaluator, although the implementation is different. For the JAX-WS run time, use the administrative console to select the **Use identity assertion** option on the caller binding panel. This defines the trusted identity token type, and then defines a list of one or more trusted identities. The trusted ID evaluator validates the trusted identity token against the list of trusted identities. For more information about the list of trusted identities, read the topic Changing the order of the callers for a token or message part.

For WebSphere Application Server Version 6.1 and later, the Caller and TrustMethod elements are used to support the requestor login. The requestor sends a message to an intermediary, and the message is dispatched to the service. Based on the security information, the service performs a login for the requestor. In some cases, there are multiple security tokens, so the service has to decide which one to use. When the requestor ID is included as an ID assertion, the service can specify how to trust the intermediary. The following intermediary scenarios are supported:

<BasicAuth, null, null>
The requestor username and password is used for authentication. In this case, authentication is performed with requestor properties, therefore a password is required for authentication.

<Signature, null, null>
The requestor signature is used for authentication.

<IDAssertion, Username, null>
The requestor username (without a password) is used to identify the requestor. The UsernameToken token is used as the ID assertion, therefore no password is required to accompany the username. In this case, the service trusts the intermediary unconditionally.

<IDAssertion, Username, Username>
The requestor username (without a password) is used to identify the requestor, and the username and password of the intermediary is used to authenticate the intermediary. The UsernameToken token, when used to establish trusted identity, always requires a password because the purpose of the token is to establish trust between the intermediary and the service.

<IDAssertion, Username, X509>
The requestor username (without a password) is used to identify the requestor, and the signature of the intermediary is used to authenticate the intermediary. In this case, the trusted identity for the signature of the intermediary must be established using an X.509 certificate.

<IDAssertion, X509, null>
The identity of the requestor is established using an X.509 certificate. In this case, the X.509

certificate from the requestor does not provide a signature to prove possession of the certificate, and therefore the service trusts the intermediary unconditionally.

<IDAssertion, X509, Username>
The identity of the requestor is established using an X.509 certificate, and the username and password of the intermediary is used to authenticate the intermediary. The UsernameToken token, when used to establish trusted identity, always requires a password because the purpose of the token is to establish trust between the intermediary and the service.

<IDAssertion, X509, X509>
The identity of the requestor is established using an X.509 certificate, and the signature of intermediary is used to authenticate the intermediary.

Hardware cryptographic device support for Web Services Security:

In IBM WebSphere Application Server Version 6.1 or later, Web Services Security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web Services Security.

Enabling cryptographic operations on hardware devices

You can enable cryptographic operations on hardware devices. The keys that are used can be stored in a Java keystore file; it is not necessary to store them on the hardware device. The decision to use enable cryptographic operations on hardware devices is made at the server level only, not at the application level.

If cryptographic operations on hardware device is enabled, the Web Service Security run time first attempts to use the hardware device for cryptographic operations. If the attempt to use the hardware device fails or if the algorithm is not supported by the hardware device, the run time uses a software provider from the security providers list.

Enabling this feature might improve the performance, depending on the hardware device. For more information on how to enable cryptographic operations on hardware devices, see Configuring hardware cryptographic devices for Web Services Security.

Secure keys

Cryptographic keys can be stored on the hardware cryptographic device and never leave the device. These secure keys are confined to the hardware cryptographic device for security considerations rather than performance considerations. The option to select whether to use keys that are stored in a hardware cryptographic device or a Java keystore file can be made at the application level.

If the keystore reference is specified to be a hardware device configuration, the Web Services Security run time first attempts to obtain the cryptographic algorithm from the hardware device. If the algorithm is not supported or fails, the run time uses a software provider from the security providers list.

See further information about how to enable secure keys, see Enabling cryptographic keys stored in hardware devices in Web Services Security.

Limitations

The hardware cryptographic device support for Web Services Security currently has the following limitations:

- There is no support for a web services client running as a Java Platform, Enterprise Edition (Java EE) Application Client.
- There is no support for hardware cryptographic devices on iSeries.

- Only Version 6.1 and later, Web Services Security applications can take advantage of the hardware cryptographic support.

  Note: Versions 5.*x* and 6.0.*x* Web Services Security applications can run in a Version 6.1 WebSphere Application Server, but these versions cannot take advantage of the hardware cryptographic support.

Long-term usage of session keys

You can configure WebSphere Application Server to use the hardware keystore, or you can configure the hardware acceleration card to allow the long-term usage of session keys. Session keys might be insecure.

If you are concerned about insecure session keys, configure WebSphere Application Server to use the hardware keystore. See the information about how to enable cryptographic keys that are stored in hardware devices in Web Services Security.

To configure the hardware acceleration card to allow the long-term usage of session keys, see the manufacturer's documentation for the specific hardware acceleration card. For example:

1. For the nCipher nforce 1600 server Version 2.23.6, follow the nCipher documentation instructions.
2. You can set the `CKNFAST_SECURITY_ASSURANCES_OVERRIDE=longterm` parameter in the `cknfastrc` configuration file. This configuration change eliminates the time limit that is associated with session keys.
3. Follow the documentation for Cipher to restart the nCipher server.
4. Restart WebSphere Application Server.

Default configuration:

You can use sample configurations with the administrative console for testing purposes.

The information in the following sections describes sample default bindings, sample general bindings, and samples for key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators. You can develop web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7, using the Java API for XML-Based Web Services (JAX-WS) programming model. Samples that are provided with WebSphere Application Server differ depending on which programming model you use.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Do not use these sample configurations in a production environment as they are for sample and testing purposes only. To make modifications to these sample configurations, it is recommended that you use the administrative console provided by WebSphere Application Server.

Detailed information on the sample general bindings for the JAX-WS programming model is available in the topic General sample bindings for JAX-WS applications.

Information on configuring default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators for the JAX-RPC programming model is available in the topic Default sample configurations for JAX-RPC.

General sample bindings for JAX-WS applications:

You can use sample bindings with the administrative console for testing purposes.

WebSphere Application Server Version 7.0 and later includes provider and client sample bindings for testing purposes. In the bindings, the product provides sample values for supporting tokens for different token types, such as the X.509 token, the username token, the LTPA token, and the Kerberos token. The bindings also include sample values for message protection information for token types such as X.509 and secure conversation. Both provider and client sample bindings can be applied to the applications attached with a system policy set, or application policy set, from the default local repository.

This information describes the general sample bindings for the Java API for XML-Based Web Services (JAX-WS) programming model. You can develop web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7.0 and later, using the Java API for XML-Based Web Services (JAX-WS) programming model. Sample general bindings may differ depending on which programming model you use. The following sections, describing various general sample bindings, are provided:

- "General client sample bindings"
- "Client sample bindings V2" on page 922
- "General provider sample bindings" on page 925
- "Provider sample bindings V2" on page 929

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Do not use these provider and client sample bindings in their default state in a production environment. You must modify the bindings to meet your security needs before using them in a production environment by making a copy of the bindings and then modifying the copy. For example, you must change the key and keystore settings to ensure security, and modify the binding settings to match your environment.

One set of general default bindings is shared by the applications to make application deployment easier. The default binding information for the server level can be overridden by the binding information on the application level. The product searches for binding information for an application on the application level before searching the server level.

General client sample bindings

- The sample configuration for signing information generation, called `asymmetric-signingInfoRequest`, contains the following configuration:
  - References the `gen_signkeyinfo` signing key information.
  - The part reference configuration, which contains the transform configuration using the `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
  - The signing key information, `gen_signkeyinfo`, which contains this configuration:
    - The security token reference.

- The gen_signx509token protection token asymmetric signature generator, as follows:
  - Contains the X.509 V3 Token v1.0 token type.
  - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
  - Contains the wss.generate.x509 JAAS login
- The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsigsender.ks, with these characteristics:
  - The keystore type is JKS.
  - The keystore password is client.
  - The alias name of the trusted certificate is soapca.
  - The alias name of the personal certificate is soaprequester.
  - The key password client issued by the intermediary certificate authority Int CA2, which is in turn issued by soapca.

– The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.

– The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.

- The sample configuration for signing information generation called symmetric-signingInfoRequest contains the following configuration:
  - References the gen_signsctkeyinfo signing key information.
  - The part reference configuration, which contains the transform configuration using the http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, gen_signsctkeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, as follows:
      - Contains the Secure Conversation Token Version 1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type as the local part value.
      - Contains wss.generate.sct JAAS login
    - The WS-Trust Callback Handler.
  - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information generation, called asymmetric-encryptionInfoRequest, contains the following configuration:
  - References the gen_enckeyinfo encryption key information.
  - Encryption key information, named gen_enckeyinfo, which contains this configuration:
    - The key identifier.
    - The gen_encx509token protection token asymmetric encryption generator, as follows:
      - Keystore type is JCEKS.
      - Keystore password is client.
      - Alias name of the trusted certificate is soapca.
      - Alias name of the personal certificate is bob.

        - Key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
      - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks.
  - The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information generation, called symmetric-encryptionInfoRequest, contains the following configuration:
  - References the gen_encsctkeyinfo encryption key information.
  - The encryption key information, gen_encsctkeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, which contains the following configuration:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains wss.generate.sct JAAS login.
    - The WS-Trust Callback Handler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for signing information consumption, called asymmetric-signingInfoResponse, contains the following configuration:
  - References the con_signkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_signkeyinfo, which contains the following configuration:
    - The con_signx509token protection token asymmetric signature consumer, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.consume.x509 JAAS login.
    - The X.509 Callback Handler, as follows:
      - References a certificate store named DigSigCertStore.
      - References a trusted anchor store named DigSigTrustAnchor.
  - The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information consumption, called symmetric-signingInfoResponse, contains the following configuration:
  - References the con_sctsignkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_sctsignkeyinfo, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.

        - WS-SecureConversation as the client label.
        - WS-SecureConversation as the service label.
        - Key length of 16 bytes.
        - Nonce length of 16 bytes.
    - The con_scttoken protection token consumer, as follows:
        - Contains the Secure Conversation Token v1.3 token type.
        - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
        - Contains the wss.consume.sct JAAS login.
    - The WS-SecureConversation Callback Handler.
  - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information consumption, called asymmetric-encryptionInfoResponse, which contains the following configuration:
  - References the dec_keyinfo encryption key information.
  - The encryption key information, named dec_keyinfo, which contains the following configuration:
    - The con_encx509token protection token asymmetric encryption consumer, as follows:
        - Contains the X.509 V3 Token v1.0 token type.
        - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
        - Contains the wss.consume.x509 JAAS login.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks, with the follow characteristics:
        - The keystore type is JCEKS.
        - The keystore password is client.
        - The alias name of the trusted certificate is soapca.
        - The alias name of the personal certificate is alice.
        - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
  - The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information consumption, called symmetric-encryptionInfoResponse, contains the following configuration:
  - References the dec_sctkeyinfo encryption key information.
  - The encryption key information, named dec_sctkeyinfo, contains the following configuration:
    - The derived key, as follows:
        - Requires explicit derived key token.
        - WS-SecureConversation as the client label.
        - WS-SecureConversation as the service label.
        - Key length of 16 bytes.
        - Nonce length of 16 bytes.
    - The con_scttoken protection token consumer, as follows:
        - Contains the Secure Conversation Token v1.3 token type.
        - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
        - Contains the wss.consume.sct JAAS login.

    - The WS-SecureConversation Callback Handler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for authentication token generation, called gen_signkrb5token, contains the following configuration:
  - The custom token type for the Kerberos v5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
  - The wss.generate.KRB5BST JAAS login.
  - The following custom properties:
    - com.ibm.wsspi.wssecurity.krbtoken.targetServiceName, the target Kerberos service name.
    - com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost, the host name associated with the target Kerberos service name,

      You must provide the correct values for your environment before using this configuration.
  - The custom Kerberos token callback handler. You must provide the correct values for the Kerberos client principal and password.
- The sample configuration for authentication token generation, called gen_signltpaproptoken, contains the following configuration:
  - The token type LTPA propagation token, as follows:
    - Contains LTPA_PROPAGATION for the local part value.
    - Contains http://www.ibm.com/websphere/appserver/tokentype for the Namespace URI value.
  - Contains the wss.generate.ltpaProp JAAS login.
  - Uses the LTPA token callback handler.
- The sample configuration for authentication token generation, called gen_signltpatoken, contains the following configuration:
  - The token type of LTPA Token v2.0, as follows:
    - Contains LTPA_PROPAGATION for the local part value.
    - Contains http://www.ibm.com/websphere/appserver/tokentype for the Namespace URI value.
  - The wss.generate.ltpa JAAS login.
  - The LTPA token callback handler.
- The sample configuration for authentication token generation, called gen_signunametoken, contains the following configuration:
  - The token type of Username Token v1.0, which uses http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken for the local part value.
  - The wss.generate.unt JAAS login.
  - The Username token callback handler, as follows:
    - Contains basic authentication fields. You must provide the correct values for your environment for client principal and password.
    - Contains the following custom properties:
      - com.ibm.wsspi.wssecurity.token.username.addNonce for adding the nonce value.
      - com.ibm.wsspi.wssecurity.token.username.addTimestamp for adding the time stamp value.

Client sample bindings V2

Two new general sample bindings, Client sample V2, and Provider sample V2, have been added to the product. While many of the configurations are the same as previous versions of the client sample and provider sample bindings, there are several additional, new sample configurations. To use these new bindings, create a new profile after installing the product. For more information, read the topic Configuring Kerberos policy sets and V2 general sample bindings.

- The sample configuration for signing information generation, called symmetric-KrbsignInfoRequest, contains the following configuration:
  - References the gen_reqKRBsignkeyinfo signing key information.
  - The part reference configuration, which contains the transform configuration using the http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, gen_reqKRBsignkeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_krb5token protection token generator, as follows:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ value type as the local part value.
      - Contains wss.generate.KRB5BST JAAS login
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler.
  - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information generation, called symmetric-KrbEncInfoRequest, contains the following configuration:
  - References the gen_reqKRBenckeyinfo encryption key information.
  - The encryption key information, gen_reqKRBenckeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_krb5token protection token generator, which contains the following configuration:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ value type for the local part value.
      - Contains wss.generate.KRB5BST JAAS login.
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for signing information consumption, called symmetric-KrbsignInfoResponse, contains the following configuration:
  - References the con_respKRBsignkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_respKRBsignkeyinfo, which contains the following configuration:

        - The derived key, as follows:
            - Requires explicit derived key token.
            - WS-SecureConversation as the client label.
            - WS-SecureConversation as the service label.
            - Key length of 16 bytes.
            - Nonce length of 16 bytes.
        - The con_krb5token protection token consumer, as follows:
            - Contains the Kerberos V5 GSS AP_REQ binary security token type.
            - Contains the http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ value type for the local part value.
            - Contains the wss.consume.KRB5BST JAAS login.
        - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler
    - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
    - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information consumption, called symmetric-KrbEncInfoResponse, contains the following configuration:
    - References the con_respKRBenckeyinfo encryption key information.
    - The encryption key information, named con_respKRBenckeyinfo, contains the following configuration:
        - The derived key, as follows:
            - Requires explicit derived key token.
            - WS-SecureConversation as the client label.
            - WS-SecureConversation as the service label.
            - Key length of 16 bytes.
            - Nonce length of 16 bytes.
        - The con_krb5token protection token consumer, as follows:
            - Contains the Kerberos V5 GSS AP_REQ binary security token type.
            - Contains the http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ value type for the local part value.
            - Contains the wss.consume.KRB5BST JAAS login.
        - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler.
    - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for authentication token generation, called gen_krb5token, contains the following configuration:
    - The custom token type for the Kerberos V5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
    - The wss.generate.KRB5BST JAAS login.
    - The following custom properties:
        - com.ibm.wsspi.wssecurity.krbtoken.targetServiceName, the target Kerberos service name.
        - com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost, the host name associated with the target Kerberos service name.

          Note: You must provide the correct values for your environment before using this configuration.
    - The custom Kerberos token callback handler.

      Note: You must provide the correct values for the Kerberos client principal and password.
- The sample configuration for authentication token generation, called con_krb5token, contains the following configuration:

  - The custom token type for the Kerberos V5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
  - The wss.consume.KRB5BST JAAS login.
  - The custom Kerberos token callback handler.

General provider sample bindings

- The sample configuration for signing information consumption, called asymmetric-signingInfoRequest, contains the following configuration:
  - References the con_signkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_signkeyinfo, which contains the following configuration:
    - The con_signx509token protection token asymmetric signature consumer, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.consume.x509 JAAS login.
    - The X.509 Callback Handler, as follows:
      - References a certificate store named DigSigCertStore.
      - References a trusted anchor store named DigSigTrustAnchor.
  - The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information consumption, called symmetric-signingInfoRequest, contains the following configuration:
  - References the con_sctsignkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_sctsignkeyinfo, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The con_scttoken protection token generator, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains the wss.consume.sct JAAS login.
    - The WS-SecureConversation Callback Handler.
  - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information consumption, called asymmetric-encryptionInfoRequest, contains the following configurations:
  - References the dec_keyinfo encryption key information.
  - The encryption key information, named dec_keyinfo, which contains the following configuration:
    - The con_encx509token protection token asymmetric encryption consumer, as follows:

      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` value type for the local part value.
      - Contains the `wss.consume.x509` JAAS login.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks`, with the following characteristics:
      - The keystore type is `JCEKS`.
      - The keystore password is `client`.
      - The alias name of the trusted certificate is `soapca`.
      - The alias name of the personal certificate is `bob`.
      - The key password client issued by intermediary certificate authority `Int CA2`, which is in turn issued by `soapca`.
  - The key encryption method `http://www.w3.org/2001/04/xmlenc#rsa-1_5`.
- The sample configuration for encryption information consumption, called `symmetric-encryptionInfoRequest`, contains the following configuration:
  - References the `dec_sctkeyinfo` encryption key information.
  - The encryption key information, named `dec_sctkeyinfo`, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The `con_scttoken` protection token consumer, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct` value type for the local part value.
      - Contains the `wss.consume.sct` JAAS login.
    - The WS-SecureConversation Callback Handler.
  - The data encryption method `http://www.w3.org/2001/04/xmlenc#aes128-cbc`.
- The sample configuration for signing information generation, called `asymmetric-signingInfoResponse`, contains the following configuration:
  - References the `gen_signkeyinfo` signing key information.
  - The part reference configuration, which uses the transform configuration `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
  - The signing key information, named `gen_signkeyinfo`, which contains the following configuration:
    - The security token reference.
    - The `gen_signx509token` protection token asymmetric signature generator, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` value type for the local part value.
      - Contains the `wss.generate.x509` JAAS login.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks`, with the following characteristics:
      - The keystore type is `JKS`.

        - The keystore password is client.
        - The alias name of the trusted certificate is soapca.
        - The alias name of the personal certificate is soapprovider.
        - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
  - The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information generation, called symmetric-signingInfoResponse, contains the following configuration:
  - References the gen_signsctkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named gen_signsctkeyinfo, which contains the following configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains the wss.generate.sct JAAS login.
    - The WS-Trust Callback Handler.
  - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information generation, called asymmetric-encryptionInfoResponse, contains the following configuration:
  - References the gen_enckeyinfo encryption key information.
  - The encryption key information, named gen_enckeyinfo, contains the following configuration
    - The key identifier.
    - The gen_encx509token protection token asymmetric encryption generator, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.generate.x509 JAAS login.
    - Uses X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks, with the following characteristics:
      - The keystore type is JCEKS.
      - The keystore password is client.
      - The alias name of the trusted certificate is soapca.
      - The alias name of the personal certificate is alice.
      - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.

  - The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information generation, called symmetric-encryptionInfoResponse, contains the following configuration:
  - References the gen_encsctkeyinfo encryption key information.
  - The encryption key information, named gen_encsctkeyinfo, contains the following configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains the wss.generate.sct JAAS login.
    - The WS-Trust Callback Handler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for authentication token consumption, called con_krb5token, contains the following configuration:
  - The custom token type for Kerberos v5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
  - The wss.consume.KRB5BST JAAS login.
  - The custom Kerberos token callback handler.
- The sample configuration for authentication token consumption, called con_ltpaproptoken, contains the following configuration:
  - The token type LTPA propagation token.
  - The wss.consume.ltpaProp JAAS login.
  - The LTPA token callback handler.
- The sample configuration for authentication token consumption, called con_ltpatoken, contains the following configuration:
  - The token type LTPA Token v2.0, with the following characteristics:
    - Contains LTPAv2 for the local part value.
    - Contains http://www.ibm.com/websphere/appserver/tokentype for the Namespace URI value.
  - The wss.consume.ltpa JAAS login
  - The LTPA token callback handler.
- The sample configuration for authentication token consumption, called con_unametoken, contains the following configuration:
  - Token type Username Token v1.0, which uses http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken for the local part value.
  - The wss.consume.unt JAAS login.
  - The Username token callback handler, with the following custom properties:
    - com.ibm.wsspi.wssecurity.token.username.verifyNonce for verifying the nonce value.
    - com.ibm.wsspi.wssecurity.token.username.verifyTimestamp for verifying the time stamp value.

Provider sample bindings V2

Two new general sample bindings, Client sample V2, and Provider sample V2, have been added to the product. While many of the configurations are the same as previous versions of the client sample and provider sample bindings, there are several additional, new sample configurations. To use these new bindings, create a new profile after installing the product. For more information, read the topic Configuring Kerberos policy sets and V2 general sample bindings.

- The sample configuration for signing information generation, called `symmetric-KrbsignInfoRequest`, contains the following configuration:
  - References the `con_respKRBsignkeyinfo` signing key information.
  - The part reference configuration, which contains the transform configuration using the `http://www.w3.org/2001/10/xml-exc-c14n# algorithm`.
  - The signing key information, `con_respKRBsignkeyinfo`, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The `con_krb5token` protection token consumer, as follows:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` value type as the local part value.
      - Contains `wss.consume.KRB5BST` JAAS login.
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler.
  - The signature method `http://www.w3.org/2000/09/xmldsig#hmac-sha1`.
  - The canonicalization method `http://www.w3.org/2001/10/xml-exc-c14n#`.
- The sample configuration for encryption information generation, called `symmetric-KrbEncInfoRequest`, contains the following configuration:
  - References the `con_reqKRBenckeyinfo` encryption key information.
  - The encryption key information, `con_reqKRBenckeyinfo`, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The `con_krb5token` protection token consumer, which contains the following configuration:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` value type for the local part value.
      - Contains `wss.consume.KRB5BST` JAAS login.
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler.
  - The data encryption method `http://www.w3.org/2001/04/xmlenc#aes128-cbc`.

- The sample configuration for signing information consumption, called `symmetric-KrbsignInfoResponse`, contains the following configuration:
  - References the `gen_respKRBsignkeyinfo` signing key information.
  - The part reference configuration, which uses the transform configuration `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
  - The signing key information, named `gen_respKRBsignkeyinfo`, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The `gen_krb5token` protection token generator, as follows:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` value type for the local part value.
      - Contains the `wss.generate.KRB5BST` JAAS login.
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler.
  - The signature method `http://www.w3.org/2000/09/xmldsig#hmac-sha1`.
  - The canonicalization method `http://www.w3.org/2001/10/xml-exc-c14n#`.
- The sample configuration for encryption information consumption, called `symmetric-KrbEncInfoResponse`, contains the following configuration:
  - References the `gen_respKRBenckeyinfo` encryption key information.
  - The encryption key information, named `gen_respKRBenckeyinfo`, contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The `gen_krb5token` protection token generator, as follows:
      - Contains the Kerberos V5 GSS AP_REQ binary security token type.
      - Contains the `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` value type for the local part value.
      - Contains the `wss.generate.KRB5BST` JAAS login.
    - The com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler
  - The data encryption method `http://www.w3.org/2001/04/xmlenc#aes128-cbc`.
- The sample configuration for authentication token generation, called `gen_krb5token`, contains the following configuration:
  - The custom token type for the Kerberos V5 token, which uses `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` for the local part value.
  - The `wss.generate.KRB5BST` JAAS login.
  - The custom Kerberos token callback handler.
- The sample configuration for authentication token generation, called `con_krb5token`, contains the following configuration:

  - The custom token type for the Kerberos V5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
  - The wss.consume.KRB5BST JAAS login.
  - The custom Kerberos token callback handler.

Default sample configurations for JAX-RPC:

Use sample configurations with the administrative console for testing purposes.

This information describes the sample default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators for WebSphere Application Server using the API for XML-based RPC (JAX-RPC) programming model. You can develop web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7 and later, using the Java API for XML-Based Web Services (JAX-WS) programming model. Sample default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluator may differ depending on which programming model you use.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Do not use these configurations in a production environment as they are for sample and testing purposes only. To make modifications to these sample configurations, it is recommended that you use the administrative console provided by WebSphere Application Server.

For a Web Services Security-enabled application, you must correctly configure a deployment descriptor and a binding. In WebSphere Application Server, one set of default bindings is shared by the applications to make application deployment easier. The default binding information for the server level can be overridden by the binding information on the application level. The Application Server searches for binding information for an application on the application level before searching the server level.

The following sample configurations are for WebSphere Application Server using the API for XML-based RPC (JAX-RPC) programming model.

Default generator binding

WebSphere Application Server provides a sample set of default generator bindings. The default generator bindings contain both signing information and encryption information.

The sample signing information configuration is called gen_signinfo and contains the following configurations:

- Uses the following algorithms for the gen_signinfo configuration:
  - Signature method: http://www.w3.org/2000/09/xmldsig#rsa-sha1
  - Canonicalization method: http://www.w3.org/2001/10/xml-exc-c14n#
- References the gen_signkeyinfo signing key information. The following information pertains to the gen_signkeyinfo configuration:
  - Contains a part reference configuration that is called gen_signpart. The part reference is not used in default binding. The signing information applies to all of the Integrity or Required Integrity elements

  within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the gen_signpart configuration:
    - Uses the transform configuration called transform1. The following transforms are configured for the default signing information:
      - Uses the http://www.w3.org/2001/10/xml-exc-c14n# algorithm
      - Uses the http://www.w3.org/2000/09/xmldsig#sha1 digest method
  - Uses the security token reference, which is the configured default key information.
  - Uses the SampleGeneratorSignatureKeyStoreKeyLocator key locator. For more information on this key locator, see "Sample key locators" on page 934.
  - Uses the gen_signtgen token generator, which contains the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the gen_signtgen_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type local name value.
  - Uses X.509 Callback Handler. The callback handler calls the ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks key store.
    - The key store password is client.
    - The alias name of the trusted certificate is soapca.
    - The alias name of the personal certificate is soaprequester.
    - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.

The sample encryption information configuration is called gen_encinfo and contains the following configurations:

- Uses the following algorithms for the gen_encinfo configuration:
  - Data encryption method: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - Key encryption method: http://www.w3.org/2001/04/xmlenc#rsa-1_5
- References the **gen_enckeyinfo** encryption key information. The following information pertains to the gen_enckeyinfo configuration:
  - Uses the key identifier as the default key information.
  - Contains a reference to the SampleGeneratorEncryptionKeyStoreKeyLocator key locator. For more information on this key locator, see "Sample key locators" on page 934.
  - Uses the gen_signtgen token generator, which has the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the gen_enctgen_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type local name value.
  - Uses X.509 Callback Handler. The callback handler calls the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks key store.
    - The key store password is storepass.
    - The secret key CN=Group1 has an alias name of Group1 and a key password of keypass.
    - The public key CN=Bob, O=IBM, C=US has an alias name of bob and a key password of keypass.
    - The private key CN=Alice, O=IBM, C=US has an alias name of alice and a key password of keypass.

Default consumer binding

WebSphere Application Server provides a sample set of default consumer binding. The default consumer binding contain both signing information and encryption information.

The sample signing information configuration is called con_signinfo and contains the following configurations:

- Uses the following algorithms for the con_signinfo configuration:
  - Signature method: http://www.w3.org/2000/09/xmldsig#rsa-sha1
  - Canonicalization method: http://www.w3.org/2001/10/xml-exc-c14n#
- Uses the con_signkeyinfo signing key information reference. The following information pertains to the con_signkeyinfo configuration:
  - Contains a part reference configuration that is called con_signpart. The part reference is not used in default binding. The signing information applies to all of the Integrity or RequiredIntegrity elements within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the con_signpart configuration:
    - Uses the transform configuration called reqint_body_transform1. The following transforms are configured for the default signing information:
      - Uses the http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
      - Uses the http://www.w3.org/2000/09/xmldsig#sha1 digest method.
  - Uses the security token reference, which is the configured default key information.
  - Uses the SampleX509TokenKeyLocator key locator. For more information on this key locator, see “Sample key locators” on page 934.
  - References the con_signtcon token consumer configuration. The following information pertains to the con_signtcon configuration:
    - Uses the X.509 Token Consumer, which is configured as the consumer for the default signing information.
    - Contains the signtconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST that references the following information:
    - Trust anchor: SampleClientTrustAnchor
    - Collection certificate store: SampleCollectionCertStore

The encryption information configuration is called con_encinfo and contains the following configurations:

- Uses the following algorithms for the con_encinfo configuration:
  - Data encryption method: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - Key encryption method: http://www.w3.org/2001/04/xmlenc#rsa-1_5
- References the con_enckeyinfo encryption key information. This key actually decrypts the message. The following information pertains to the con_enckeyinfo configuration:
  - Uses the key identifier, which is configured as the key information for the default encryption information.
  - Contains a reference to the SampleConsumerEncryptionKeyStoreKeyLocator key locator. For more information on this key locator, see “Sample key locators” on page 934.
  - References the con_enctcon token consumer configuration. The following information pertains to the con_enctcon configuration:
    - Uses the X.509 token consumer, which is configured for the default encryption information.
    - Contains the enctconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST.

Sample key store configurations

The following sample key stores are for testing purposes only; do not use these key stores in a production environment:

- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks
  - The key store format is JKS.
  - The key store password is client.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soaprequester alias name and a client key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks
  - The key store format is JKS.
  - The key store password is server.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soapprovider alias name and a server key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks
  - The key store format is JCEKS.
  - The key store password is storepass.
  - The CN=Group1 DES secret key has a Group1 alias name and a keypass key password.
  - The CN=Bob, O=IBM, C=US public key has a bob alias name and a keypass key password.
  - The CN=Alice, O=IBM, C=US private key has a alice alias name and a keypass key password.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks
  - The key store format is JCEKS.
  - The key store password is storepass.
  - The CN=Group1 DES secret key has a Group1 alias name and a keypass key password.
  - The CN=Bob, O=IBM, C=US private key has a bob alias name and a keypass key password.
  - The CN=Alice, O=IBM, C=US public key has a alice alias name and a keypass key password.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
  - The intermediary certificate is signed by soapca and it signs both the soaprequester and the soapprovider.

Sample key locators

Key locators are used to locate the key for digital signature, encryption, and decryption. For information on how to modify these sample key locator configurations, see the following articles:

- Configuring the key locator using JAX-RPC for the generator binding on the application level
- Configuring the key locator using JAX-RPC for the consumer binding on the application level
- Configuring the key locator using JAX-RPC on the server level

SampleClientSignerKey

This key locator is used by the request sender for a Version 5.x application to sign the SOAP message. The signing key name is clientsignerkey, which is referenced in the signing information as the signing key name.

SampleServerSignerKey

This key locator is used by the response sender for a Version 5.x application to sign the SOAP message. The signing key name is serversignerkey, which can be referenced in the signing information as the signing key name.

SampleSenderEncryptionKeyLocator
This key locator is used by the sender for a Version 5.x application to encrypt the SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` key store and the `com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator` key store key locator. The implementation is configured for the DES secret key. To use asymmetric encryption (RSA), you must add the appropriate RSA keys.

SampleReceiverEncryptionKeyLocator
This key locator is used by the receiver for a Version 5.x application to decrypt the encrypted SOAP message. The implementation is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` key store and the `com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator` key store key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). To use RSA, you must add the private key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`.

SampleResponseSenderEncryptionKeyLocator
This key locator is used by the response sender for a Version 5.x application to encrypt the SOAP response message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` key store and the `com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator` key store key locator. This key locator maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key `alice`, and you must change WebSphere Application Server to the appropriate user. The `SampleResponseSenderEncryptionKeyLocator` key locator also can set a default key for encryption. By default, this key locator is configured to use public key `alice`.

SampleGeneratorSignatureKeyStoreKeyLocator
This key locator is used by generator to sign the SOAP message. The signing key name is `SOAPRequester`, which is referenced in the signing information as the signing key name. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store and the `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator` key store key locator.

SampleConsumerSignatureKeyStoreKeyLocator
This key locator is used by the consumer to verify the digital signature in the SOAP message. The signing key is `SOAPProvider`, which is referenced in the signing information. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks` key store and the `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator` key store key locator.

SampleGeneratorEncryptionKeyStoreKeyLocator
This key locator is used by the generator to encrypt the SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` key store and the `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator` key store key locator.

SampleConsumerEncryptionKeyStoreKeyLocator
This key locator is used by the consumer to decrypt an encrypted SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` key store and the `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator` key store key locator.

SampleX509TokenKeyLocator
This key locator is used by the consumer to verify a digital certificate in an X.509 certificate. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` key store and the `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator` key store key locator.

Sample collection certificate store

Collection certificate stores are used to validate the certificate path. For information on how to modify this sample collection certificate store, see the following articles:

- Configuring the collection certificate store for the generator binding on the application level
- Configuring the collection certificate store for the consumer binding on the application level

- Configuring the collection certificate on the server level

SampleCollectionCertStore
This collection certificate store is used by the response consumer and the request generator to validate the signer certificate path.

Sample trust anchors

Trust anchors are used to validate the trust of the signer certificate. For information on how to modify the sample trust anchor configurations, see the following articles:

- Configuring trust anchors for the generator binding on the application level
- Configuring trust anchors for the consumer binding on the application level
- Configuring trust anchors on the server level

SampleClientTrustAnchor
This trust anchor is used by the response consumer to validate the signer certificate. This trust anchor is configure to access the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store.

SampleServerTrustAnchor
This trust anchor is used by the request consumer to validate the signer certificate. This trust anchor is configure to access the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store.

Sample trusted ID evaluators

Trusted ID evaluators are used to establish trust before asserting the identity in identity assertion. For information on how to modify the sample trusted ID evaluator configuration, see Configuring trusted ID evaluators on the server level.

SampleTrustedIDEvaluator
This trusted ID evaluator uses the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl implementation. The default implementation of com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator contains a list of trusted identities. This list, which is used for identity assertion, defines the key name and value pair for the trusted identity. The key name is in the form trustedId_* and the value is the trusted identity. For more information, see the example in Configuring trusted ID evaluators on the server level.

Complete the following steps to define this information for the server level in the administrative console:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using WebSphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web Services Security.**
3. Under Additional properties, click **Trusted ID evaluators** > **SampleTrustedIDEvaluator**.

Default implementations of the Web Services Security service provider programming interfaces:

This information describes the default implementations of the service provider interfaces (SPI) for Web Services Security within WebSphere Application Server. The default implementation classes and their functionality for both the JAX-RPC run time and the JAX-WS run time are discussed. You can use this information to create or modify the Web Services Security binding configuration.

best-practices: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new web services applications and clients.

Default implementations for the JAX-RPC run time

com.ibm.wsspi.wssecurity.token.X509TokenGenerator

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. It is responsible for creating the X.509 token object from the X.509 certificate, which is returned by the com.ibm.wsspi.wssecurity.auth.callback.{X509,PKCS7,PkiPath}CallbackHandler interface. Encode the token using the base 64 format and insert its XML representation into the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface and it retrieves the X.509 certificate from the keystore file.

com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. It is responsible for creating the username token object from user name and password that is returned by a javax.security.auth.callback.CallbackHandler implementation such as the following callback handler:

com.ibm.wsspi.wssecurity.auth.callback{GUIPrompt,NonPrompt,StdinPrompt}CallbackHandler.

It also inserts the XML representation of the token into the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class retrieves the keys from the keystore files for digital signature and encryption.

com.ibm.wsspi.wssecurity.token.X509TokenConsumer

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the X.509 token from the binary security token. This class decodes the Base64 encryption within the X.509 token and then invokes the system.wssecurity.X509BST Java Authentication and Authorization Service (JAAS) Login Configuration with the com.ibm.wsspi.wssecurity.auth.module.X509LoginModule login module to validate the X.509 token. An object of the com.ibm.wsspi.wssecurity.auth.token.X509Token is created for the validated X.509 token and stored in JAAS Subject.

com.ibm.wsspi.wssecurity.token.IDAssertionUsernameTokenConsumer

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the username token for identity assertion (IDAssertion), which does not have a password element. This interface invokes the system.wssecurity.IDAssertionUsernameToken JAAS login configuration with the com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule login module to validate the IDAssertion user name token. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule
This class implements the javax.security.auth.spi.LoginModule interface and checks whether the username value is not empty. The login module assumes that the UsernameToken is valid if the username value is not empty.

com.ibm.wsspi.wssecurity.token.LTPATokenGenerator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. This class is responsible for Base 64 encoding the LTPA token object obtained from the com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler callback handler. The object is inserted into the Web Services Security header within the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.token.LTPATokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the LTPA token from the binary security token, and decodes the Base64 encoding within the LTPA token. An object of the com.ibm.wsspi.wssecurity.auth.token.LTPAToken class is created for the validated LTPA token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.auth.module.X509LoginModule
This class implements the javax.security.auth.spi.LoginModule interface and validates the X.509 Certificate based on the trust anchor and the collection certification store configuration.

com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the username token, extracts the user name and password, and then invokes the system.wssecurity.UsernameToken JAAS login configuration using the com.ibm.wsspi.wssecurity.auth.module.UsernameLoginModule login module to validate the user name and password. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class is used to retrieve a public key from a X.509 certificate. The X.509 certificate is stored in the X.509 token (com.ibm.wsspi.wssecurity.auth.token.X509Token) in the JAAS Subject. The X.509 token is created by the X.509 Token Consumer (com.ibm.wsspi.wssecurity.tokenX509TokenConsumer).

com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class is used to retrieve a public key from the X.509 certificate of the request signer and encrypt the response. You can use this key locator in the response generator binding configuration only.

Important: This implementation assumes that only one signer certificate is used in the request.

com.ibm.wsspi.wssecurity.auth.token.UsernameToken
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the username token.

com.ibm.wsspi.wssecurity.auth.token.X509Token
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the X.509 binary security token (X.509 certificate).

com.ibm.wsspi.wssecurity.auth.token.LTPAToken
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class as a wrapper to the LTPA token that is extracted from the binary security token.

com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface and is responsible for creating a certificate and binary data with or without a certificate revocation list (CRL) using the PKCS#7 encoding. The certificate and the binary data is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate and binary data without a CRL using the PkiPath encoding. The certificate and binary data is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate from the keystore file. The X.509 token certificate is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler

This implementation generates a Lightweight Third Party Authentication (LTPA) token in the Web Services Security header as a binary security token. If basic authentication data is defined in the application binding file, it is used to perform a login, to extract the LTPA token from the WebSphere Application Server credentials, and to insert the token in the Web Services Security header. Otherwise, it extracts the LTPA security token from the invocation credentials (run as identity) and inserts the token in the Web Services Security header.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

This implementation reads the basic authentication data from the application binding file. You might use this implementation on the server side to generate a username token.

com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler

This implementation presents you with a login prompt to gather the basic authentication data. Use this implementation on the client side only.

com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler

This implementation collects the basic authentication data using a standard in (stdin) prompt. Use this implementation on the client side only.

Restriction: If you have a multi-threaded client and multiple threads attempt to read from standard in at the same time, all the threads will not successfully obtain the user name and password information. Therefore, you cannot use the com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler implementation with a multi-threaded client where multiple threads might attempt to obtain data from standard in concurrently.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator

This interface is used to evaluate the level of trust for identity assertion. The default implementation is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, which enables you to define a list of trusted identities.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl

This default implementation enables you to define a list of trusted identities for identity assertion.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorException

This exception class is used by an implementation of the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator to communicate the exception and errors to the Web Services Security run time.

Default implementations for the JAX-WS run time

com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenGenerator
This implementation invokes the JAAS CallbackHandler and JAAS login configuration that are specified in the binding to create the SecurityToken at run time on the outbound SOAP message.

com.ibm.websphere.wssecurity.callbackhandler.X509GenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and retrieves the X.509 certificate. The following properties may be specified:

- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed. This property takes a boolean value, and the default value is `false`.
- com.ibm.wsspi.wssecurity.token.cert.useRequestorCert. This property takes a boolean value, and the default value is `false`.

com.ibm.ws.wssecurity.wssapi.token.impl.X509GenerateLoginModule
The wss.generate.x509 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.X509GenerateLoginModule. X509GenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML Username token structure, and also a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7GenerateLoginModule
The wss.generate.pkcs7 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7GenerateLoginModule. PKCS7GenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathGenerateLoginModule
The wss.generate.pkiPath JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathGenerateLoginModule. PkiPathGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the binding configuration and user name and password authentication data. The following properties may be specified. These properties take a boolean value, and the default value is `false`.

- com.ibm.wsspi.wssecurity.token.username.addNonce
- com.ibm.wsspi.wssecurity.token.username.addTimestamp
- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed
- com.ibm.wsspi.wssecurity.token.IDAssertion.useRunAsIdentity
- com.ibm.wsspi.wssecurity.token.IDAssertion.sendRealm
- com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm

com.ibm.ws.wssecurity.wssapi.token.impl.UNTGenerateLoginModule
The wss.generate.unt JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl. UNTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML Username token structure and also a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has a the value of `true`, the generated username token does not contain a password. When com.ibm.wsspi.wssecurity.token.IDAssertion.sendRealm has the value of `true`, the user name is qualified by the local realm name. When com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm

has the value of `true`, the user name field contains both the user name and a registry-dependent unique identifier for the user. Both the user name and the unique identifier are qualified by the local realm name.

com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the Kerberos user name and password, along with other binding configuration properties. The following properties may be specified. The properties take a string that specifies the target service name as part of a service principal name (SPN), in the form of `service_name/host_name@Kerberos_realm_name`.

- com.ibm.wsspi.wssecurity.krbtoken.targetServiceName
- com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost
- com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm

com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule
The wss.generate.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. KRBGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule
The wss.generate.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. DKTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time when the **Requires derived keys** option is enabled.

com.ibm.websphere.wssecurity.callbackhandler.LTPAGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the user name and password binding data if they are specified.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAGenerateLoginModule
The wss.generate.ltpa JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAGenerateLoginModule. LTPAGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. The security token contains an LTPA token that is generated from the user name and password if they are defined in the binding data, or the LTPA authentication token from the RunAs Subject, in that order.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationGenerateLoginModule
The wss.generate.ltpaProp JAAS system login configuration contains com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationGenerateLoginModule. LTPAPropagationGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. The security token contains the serialized RunAs Subject.

com.ibm.ws.wssecurity.impl.auth.callback.WSTrustCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves security context token configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule
The wss.generate.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. SCTGenerateLoginModule

implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the security context token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule

The wss.generate.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. DKTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time when the **Requires derived keys** option is enabled.

com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer

This implementation invokes the JAAS CallbackHandler and JAAS login configuration that are specified in the binding to extract the security token from the inbound SOAP message and to create the SecurityToken object at run time.

com.ibm.websphere.wssecurity.callbackhandler.X509ConsumeCallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface on SOAP message inbound to retrieve the trust store and certificate file information that are required to validate the X.509 certificate.

com.ibm.ws.wssecurity.wssapi.token.impl.X509ConsumeLoginModule

The wss.consume.x509 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.X509ConsumeLoginModule. X509ConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7ConsumeLoginModule

The wss.consume.pkcs7 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7ConsumeLoginModule PKCS7ConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathConsumeLoginModule

The wss.consume.pkiPath JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathConsumeLoginModule. PkiPathConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.websphere.wssecurity.callbackhandler.UNTConsumeCallbackHandler

This class implements the javax.security.auth.callback.CallbackHandler interface on SOAP message inbound to retrieve binding configuration data. The following properties may be specified. These properties take a boolean value and the default value is `false`.

- com.ibm.wsspi.wssecurity.token.username.verifyTimestamp
- com.ibm.wsspi.wssecurity.token.username.verifyNonce
- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed
- com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm
- com.ibm.wsspi.wssecurity.token.UsernameToken.disableUserRegistryCheck

com.ibm.ws.wssecurity.wssapi.token.impl.UNTConsumeLoginModule

The wss.consume.unt JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.UNTConsumeLoginModule. UNTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the username token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the username token at run time. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has the value of `false`, UNTConsumeLoginModule validates the username and password against the local user registry. An incorrect user name or incorrect or missing password will cause the token validation to fail. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has a value of `true`, and com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm has a value of `false`, the user name is validated against the local user registry. There should be no password in the username token. When both com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed and com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm have a value of `true`, the user name field must contain a realm-qualified user name and unique user identifier data, and the realm must be one of the trusted realms in the multiple security domain inbound trust configuration.

com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the inbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsumeLoginModule
The wss.consume.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. KRBConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the Kerberos AP_REQ token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the AP_REQ token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule
The wss.consume.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. DKTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving the derived key when a derived key is required.

com.ibm.websphere.wssecurity.callbackhandler.LTPAConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the inbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAConsumeLoginModule
The wss.consume.ltpa JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAConsumeLoginModule. LTPAConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the LTPA v2 or LTPA token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the LTPA v2 or LTPA token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationConsumeLoginModule
The wss.consume.ltpaProp JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationConsumeLoginModule. LTPAPropagationConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving, deserializing, and validating the propagation token and reconstructing the security context.

com.ibm.ws.wssecurity.impl.auth.callback.SCTConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule
The wss.consume.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule, and

com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. SCTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the security context token.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule

The wss.consume.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. DKTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving the derived key when a derived key is required.

com.ibm.ws.wssecurity.impl.auth.module.PreCallerLoginModule

The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.PreCallerLoginModule. PreCallerLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for validating whether it has received any security token that may be used to establish caller identity or trusted identity.

com.ibm.ws.wssecurity.impl.auth.module.UNTCallerLoginModule

The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.UNTCallerLoginModule. UNTCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. UNTCallerLoginModule also determines if the user identity is authorized to make an identity assertion if the username is configured to be a trusted identity, or if there is exactly one caller identity if the username token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.X509CallerLoginModule

The wss.caller JAAS system login configuration contains com.ibm.ws.wssecurity.impl.auth.module.X509CallerLoginModule. X509CallerLoginModule implements the javax.security.auth.spi.LoginModule interface. X509CallerLoginModule checks to see if the user identity is authorized to make an identity assertion if the X509 token is configured to be a trusted identity, or if there is exactly one caller identity if the X509 token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.LTPACallerLoginModule

The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.LTPACallerLoginModule. LTPACallerLoginModule implements the javax.security.auth.spi.LoginModule interface. LTPACallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the LTPA token is configured to be a trusted identity, or if there is exactly one caller identity if the LTPA token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.LTPAPropagationCallerLoginModule

The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.LTPAPropagationCallerLoginModule. LTPAPropagationCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. LTPAPropagationCallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the propagation token is configured to be a trusted identity, or if there is exactly one caller identity if the propagation token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.KRBCallerLoginModule

The wss.caller JAAS system login configuration contains com.ibm.ws.wssecurity.impl.auth.module.KRBCallerLoginModule. KRBCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. KRBCallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the Kerberos token is configured to be a trusted identity, or if there is exactly one caller identity if the Kerberos token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.WSWSSLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.WSWSSLoginModule. WSWSSLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for asserting the caller identity to the ltpaLoginModule and the wsMapDefaultInboundLoginModule to establish the caller security context.

com.ibm.ws.security.server.lm.ltpaLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.security.server.lm.ltpaLoginModule.

com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule.

XML digital signature

XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

XML digital signature does not introduce new cryptographic algorithms. WebSphere Application Server uses XML digital signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method.

XML canonicalization (c14n) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. For example, although their octet representations are different, the following examples are identical:

- `<person first="John" last="Smith"/>`
- `<person last="Smith" first="John"></person>`

C14n is a process that is used to canonicalize XML information. Select an appropriate c14n algorithm because the information that is canonicalized is dependent upon this algorithm. One of the major c14n algorithms, Exclusive XML Canonicalization, canonicalizes the character encoding scheme, attribute order, namespace declarations, and so on. The algorithm does not canonicalize white space outside tags, namespace prefixes, or data type representation.

XML signature in the Web Services Security-Core specification

The Web Services Security-Core (WSS-Core) specification defines a standard way for SOAP messages to incorporate an XML signature. You can use almost all of the XML signature features in WSS-Core except enveloped signature and enveloping signature. However, WSS-Core has some recommendations such as exclusive canonicalization for the c14n algorithm and some additional features such as SecurityTokenReference and KeyIdentifier.

The KeyIdentifier is the value of the SubjectKeyIdentifier field within the X.509 certificate. For more information on the KeyIdentifier, see “Reference to a Subject Key Identifier” within the OASIS Web Services Security X.509 Certificate Token Profile documentation.

By including XML signature in SOAP messages, the following issues are realized:

Message integrity
A message receiver can confirm that attackers or accidents have not altered parts of the message after these parts are signed by a key.

Authentication

You can assume that a valid signature is proof of possession. A message with a digital certificate that is issued by a certificate authority and a signature in the message that is validated successfully by a public key in the certificate, is proof that the signer has the corresponding private key. The receiver can authenticate the signer by checking the trustworthiness of the certificate.

Collection certificate store

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

A collection certificate store is used when WebSphere Application Server is processing a received SOAP message. For JAX-RPC applications, this collection is configured in the Request Consumer Service Configuration Details section of the binding file for servers and in the Response Consumer Configuration section of the binding file for clients. You can configure these two sections using one of the assembly tools provided by WebSphere Application Server. See the assembly tools information in the topic Assembly tools.

For JAX-WS applications, this collection is configured using the administrative console in the Keys and certificates panel of the WS-Security policy set bindings.

A collection certificate store is one kind of certificate store. A certificate store is defined as javax.security.cert.CertStore in the Java CertPath application programming interface (API). The Java CertPath API defines the following types of certificate stores:

Collection certificate store

A collection certificate store accepts the certificates and CRLs as Java collection objects.

Lightweight Directory Access Protocol certificate store

The Lightweight Directory Access Protocol (LDAP) certificate store accepts certificates and CRLs as LDAP entries.

The CertPath API uses the certificate store and the trust anchor to validate the incoming X.509 certificate that is embedded in the SOAP message. The Web Services Security implementation in the WebSphere Application Server supports the collection certificate store. Each certificate and CRL is passed as an encoded file.

Certificate revocation list

A *certificate revocation list* is a time-stamped list of certificates that have been revoked by a certificate authority (CA).

A certificate that is found in a certificate revocation list (CRL) might not be expired, but is no longer trusted by the certificate authority that issued the certificate. The certificate authority creates the CRL that contains the serial number and issuing CA distinguished name of the certificate that has been revoked. The CA might add the certificate to the certificate revocation list if it believes that the client certificate is compromised. The certificate revocation list is maintained and issued by the certificate authority.

XML encryption

XML encryption is a specification that was developed by World Wide Web (WWW) Consortium (W3C) in 2002 and that contains the steps to encrypt data, the steps to decrypt encrypted data, the XML syntax to represent encrypted data, the information to be used to decrypt the data, and a list of encryption algorithms, such as triple DES, AES, and RSA.

You can apply XML encryption to an XML element, XML element content, and arbitrary data, including an XML document. For example, suppose that you need to encrypt the <CreditCard> element that is shown in example 1.

Example 1: Sample XML document:

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Example 2: XML document with a common secret key:

Example 2 shows the XML document after encryption. The <EncryptedData> element represents the encrypted <CreditCard> element. The <EncryptionMethod> element describes the applied encryption algorithm, which is triple DES in this example. The <KeyInfo> element contains the information that is needed to retrieve a decryption key, which is a <KeyName> element in this example. The <CipherValue> element contains the cipher text that is obtained by serializing and encrypting the <CreditCard> element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
       Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
       <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
         <KeyName>John Smith</KeyName>
       </KeyInfo>
       <CipherData>
         <CipherValue>ydUNqHkMrD...</CipherValue>
       </CipherData>
  </EncryptedData>
</PaymentInfo>
```

Example 3: XML document encrypted with the public key of the recipient:

In example 2, it is assumed that both the sender and recipient have a common secret key. If the recipient has a public and private key pair, which is commonly the case, the <CreditCard> element can be encrypted as shown in example 3. The <EncryptedData> element is the same as the <EncryptedData> element found in Example 2. However, the <KeyInfo> element contains an <EncryptedKey> element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XML Encryption in the WSS-Core:

The WSS-Core specification is under development by Organization for the Advancement of Structured Information Standards (OASIS). The specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The message confidentiality is realized by encryption based on XML Encryption.

The WSS-Core specification supports encryption of any combination of body blocks, header blocks, their substructures, and attachments of a SOAP message. When you encrypt parts of a SOAP message, the specification also requires that you prepend a reference from the security header block to the encrypted parts of the message. The reference can be a clue for a recipient to identify which encrypted parts of the message to decrypt.

The XML syntax of the reference varies according to what information is encrypted and how it is encrypted. For example, suppose that the <CreditCard> element in example 4 is encrypted with either a common secret key or the public key of the recipient.

Example 4: Sample SOAP Version 1.1 message:

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` so the example changes to the following:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The resulting SOAP messages are shown in Examples 5 and 6. In these example, the <ReferenceList> and <EncryptedKey> elements are used as references, respectively.

Example 5: SOAP Version 1.1 message encrypted with a common secret key

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` and the example changes to the following:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 6: SOAP message encrypted with the public key of the recipient:

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` and the example changes to the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Relationship to digital signature:

The WSS-Core specification also provides message integrity, which is realized by a digital signature that is based on the XML-Signature specification.

A combination of encryption and digital signature over common data introduces cryptographic vulnerabilities.

Symmetric versus asymmetric encryption

For XML encryption, the application server supports two types of encryption:

- *Symmetric encryption*

  In releases of the application server prior to WebSphere Application Server Version 7, including the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services, by default the KeyName reference was used to refer to the shared key outside of the SOAP message. However, the Web Services Security (WS-Security) Version 1.1 standard does not recommend using the KeyName reference. Because KeyName is not supported by the security policy, it is not supported in the application server.

  The Web Services Secure Conversation (WS-SecureConversation) standard defines how to exchange the shared key between the client and the service and how to refer to the shared key in the message. The use of Kerberos with Web Services Security, as described in the Kerberos Token Profile, also defines how to use a Kerberos session key or key derived from the session key to perform symmetric encryption. Therefore, you can use symmetric encryption by using WS-SecureConversation or Kerberos. WebSphere Application Server supports DerivedKeyToken when using WS-SecureConversation. When using Kerberos, WebSphere Application Server supports both the use of DerivedKeyToken and the use of the Kerberos session key directly.
- *Asymmetric encryption*

  For asymmetric encryption, XML Encryption introduces the idea of key wrapping. The data, such as the contents of the SOAP body element, is encrypted with a shared key that is dynamically generated while processing. Then, the generated shared key is encrypted with the public key of the receiver. WebSphere Application Server supports the X509Token for asymmetric encryption.

Security token

Web Services Security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

A specific type of security token is not required by Web Services Security. Web Services Security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification. However, the security token usage for Web Services Security is defined in separate profiles such as the Username token profile, the X.509 token profile, the Security Assertion Markup Language (SAML) token profile, the eXtensible rights Markup Language (XrML) token profile, the Kerberos token profile, and so on.

A security token is embedded in the SOAP message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the WebSphere Application Server Web Services Security handler authenticates the security token and sets up the caller identity on the running thread.

WebSphere Application Server contains an enhanced security token that has the following features:

- The client can send multiple tokens to downstream servers.
- The receiver can determine which security token to use for authorization based upon the type or signed part for X.509 tokens.
- You can use the custom token or derived key token for digital signing or encryption.

LTPA and LTPA Version 2 tokens

Web services security supports both LTPA (Version 1) and LTPA Version 2 (LTPA2) tokens. The LTPA2 token, which is more secure than Version 1, is supported by the JAX-WS runtime only.

Note: The support statements in this topic apply to the web services security implementation for WebSphere Application Server and not the security implementation for non-web services functionality.

The Lightweight Third Party Authentication (LTPA) token is a specific type of binary security token. The web services security implementation for WebSphere Application Server, Version 5 and later supports the LTPA Version 1 token. WebSphere Application Server Version 7 and later supports the LTPA Version 2 token using the JAX-WS runtime environment.

Although the same LTPAToken assertion is used in the policy for both LTPA Version 1 and LTPA Version 2, the valuetype value for the Version 2 token is different than Version 1. The valuetype value is composed of the URI and the local name. The following table shows the valuetype values for the LTPA token versions when they are selected as the token type for the policy set bindings. These values are not editable.

Table 82. LTPA token versions and their valuetype values. This table lists the valuetype values for both LTPA (Version 1) and LTPA2 tokens.

LTPA Version token	Valuetype value
LTPA (Version 1)	`http://www.ibm.com/websphere/appserver/tokentype/5.0.2/LTPA`
LTPA2	`http://www.ibm.com/websphere/appserver/tokentype/LTPAv2`

To allow for interoperability between servers that are running different versions of WebSphere Application Server, by default, the JAX-WS web services security runtime in Version 7.0 and later can successfully consume an LTPA Version 1 token when the binding is configured to expect an LTPA2 token. However, you can configure the binding for the JAX-WS runtime to accept only LTPA2 tokens. For more information, see the documentation about Authentication generator or consumer token settings.

If the web services security run time receives a token with a unrecognized valuetype value and the SOAP security header contains a mustUnderstand attribute value that is equal to `'1'`, the web services security run time issues a SOAPFaultException error. If the mustUnderstand attribute value is equal to `'0'`, the token is ignored.

If an LTPA2 token is sent with a mustUnderstand attribute value that is equal to `'1'` to a web services security run time in which the LTPA2 token is not supported, the run time does not recognize the LTPAv2 valuetype value. Thus, the receiving run time issues a SOAPFaultException error. The following table illustrates these different configurations and their potential error messages..

Table 83. LTPA token configurations. This table lists whether the LTPA Version 1 token is optional or required, lists the associated mustUnderstand attribute value, lists its run time, and provides the resulting SOAPFaultException error, if applicable

Run time	LTPA Version 1 token status	MustUnderstand attribute value	SOAPFaultException error
JAX-RPC	Required	1	`com.ibm.wsspi.wssecurity.SoapSecurityException: WSEC5509E: A security token whose type is [{http://www.ibm.com/websphere/appserver/tokentype/5.0.2}LTPA] is required.`
JAX-RPC	Required	0	`com.ibm.wsspi.wssecurity.SoapSecurityException: WSEC5509E: A security token whose type is [{http://www.ibm.com/websphere/appserver/tokentype/5.0.2}LTPA] is required.`
JAX-RPC	Optional	1	`com.ibm.wsspi.wssecurity.SoapSecurityException: WSEC5502E: Unexpected element as the target element: s:BinarySecurityToken.`
JAX-RPC	Optional	0	None
JAX-RPC	Not Configured	1	`com.ibm.wsspi.wssecurity.SoapSecurityException: WSEC5502E: Unexpected element as the target element: s:BinarySecurityToken.`
JAX-RPC	Not Configured	0	None
JAX-WS (Version 6.1 Feature Pack for Web Services)	Not Configured	1	`CWWSS5502E: The target element: s:BinarySecurityToken was not expected.`
JAX-WS (Version 6.1 Feature Pack for Web Services)	Not Configured	0	None
JAX-WS (Version 6.1 Feature Pack for Web Services)	Configured	1	`CWWSS5509E: A security token whose type is [{http://www.ibm.com/websphere/appserver/tokentype/5.0.2}LTPA] is required.`
JAX-WS (Version 6.1 Feature Pack for Web Services)	Configured	0	`CWWSS5509E: A security token whose type is [{http://www.ibm.com/websphere/appserver/tokentype/5.0.2}LTPA] is required.`

You can configure the JAX-WS run time to generate either LTPA (Version 1) or LTPA2 tokens. If you configure the LTPA token generator in a policy binding to generate an LTPA (Version 1) token, you must do one of the following:

- Enable the single sign-on interoperability mode, which is available on the Single sign-on (SSO) panel within the administrative console. For more information on this option, see the documentation about single sign-on settings.
- Set the `com.ibm.wsspi.wssecurity.tokenGenerator.ltpav1.pre.v7` custom property to true for the LTPA token generator.

If you do not perform at least one of the steps previously indicated, an error occurs when the application, which is attached to these bindings, is started.

Username token

You can use the <UsernameToken> element to propagate a user name and, optionally, password information. Also, you can use this token type to carry basic authentication information. Both a user name and a password are used to authenticate the SOAP message.

OASIS: Web Services Security UsernameToken Profile 1.0

A UsernameToken element containing the user name is used in identity assertion. Identity assertion establishes the identity of the user based on the trust relationship.

The following example shows the syntax of the <UsernameToken> element:

```
<wsse:UsernameToken wsu:Id="Example-1">
   <wsse:Username>
   ...
   </wsse:Username>
   <wsse:Password Type="...">
   ...
   </wsse:Password>
   <wsse:Nonce EncodingType="...">
   ...
   </wsse:Nonce>
   <wsu:Created>
   ...
   </wsu:Created>
</wsse:UsernameToken>
```

The Web Services Security specification defines the following password types:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText (default)
This type is the actual password for the user name.

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest
The type is the digest of the password for the user name. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

WebSphere Application Server supports the default PasswordText type. However, it does not support password digest because most user registry security policies do not expose the password to the application software.

The following example illustrates the use of the <UsernameToken> element:

```
<S:Envelope
       xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <S:Header>
           ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Joe</wsse:Username>
                <wsse:Password>ILoveJava</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
</S:Envelope>
```

OASIS: Web Services Security UsernameToken Profile 1.1

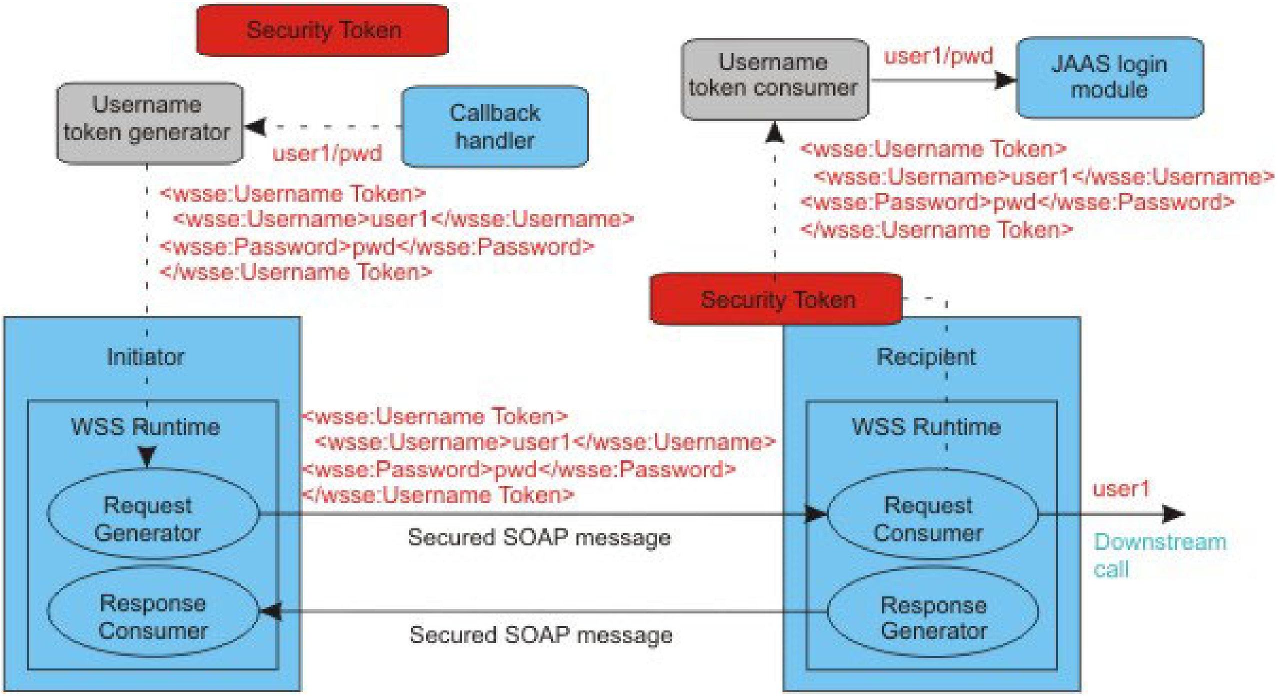
WebSphere Application Server supports both Username Token Profile 1.0 and Version 1.1 standards.

WebSphere Application Server does not support the following functions:

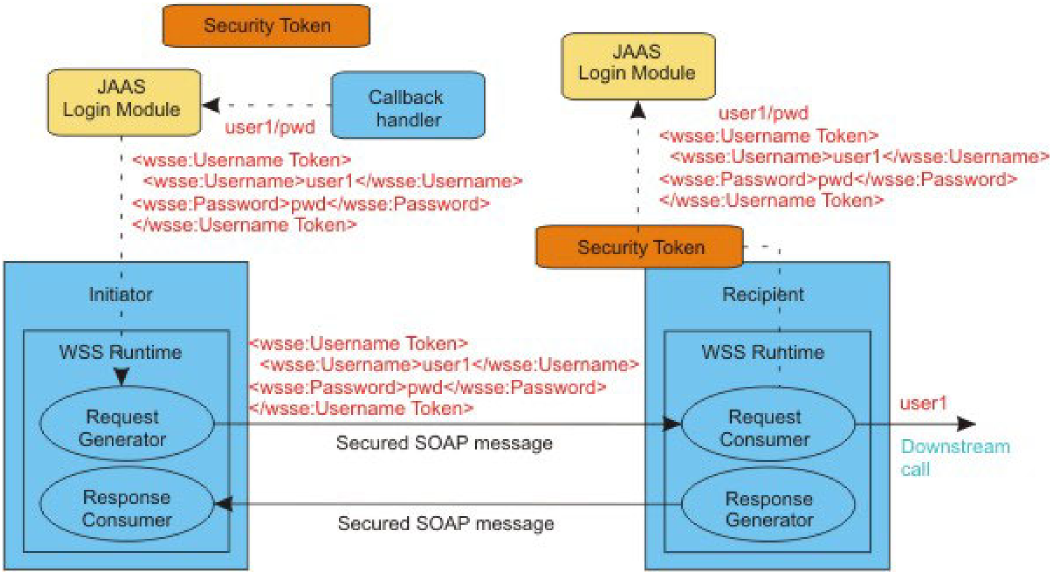
- In both versions of the Username Token Profile specification, the digest password type is not supported
- In both versions of the Username Token Profile specification, key derivation based on a password is not supported.

You can use policy sets to configure the UsernameToken using the administrative console. Also, you can use the Web Services Security APIs to attach the Username token to the SOAP message. The following figure describes the creation and validation of the Username token for the JAX-RPC and the JAX-WS programming models.

Creating and validating the Username token using the JAAS Login Module and the JAAS CallbackHandler in JAX-RPC



Creating and validating the Username token using the JAAS Login Module and the JAAS CallbackHandler in JAX-WS



Note: The WSS API is available only when you are using the Java API for XML-Based Web Services (JAX-WS) programming model.

On the generator side, the Username token is created by using the JAAS LoginModule and by using the JAAS CallbackHandler to pass the authentication data. The JAAS LoginModule creates the UsernameToken object and passes it to the Web Services Security run time.

On the consumer side, the Username Token XML format is passed to the JAAS LoginModule for validation or authentication, and the JAAS CallbackHandler is used to pass the authentication data from the Web Services Security run time to the JAAS LoginModule. After the token is authenticated, a UsernameToken object is created and is passed to the Web Service Security run time.

The following example provides sample code for creating Username tokens:

```
WSSFactory factory = WSSFactory.getInstance();
   WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Attach the username token to the message.
   UNTGenerationCallbackHandler ugCallbackHandler =
      newUNTGenerationCallbackHandler("alice", "ecila");
   SecurityToken ut = factory.newSecurityToken(ugCallbackHandler,
                                                UsernameToken.class);
   gencont.add(ut);

// Generate the WS-Security header
gencont.process(msgctx);
```

XML token

XML tokens are offered in two well-known formats called Security Assertion Markup Language (SAML) and eXtensible rights Markup Language (XrML).

In WebSphere Application Server Versions 6 and later, you can plug in your own implementation. By using extensibility of the `<wsse:Security>` header in XML-based security tokens, you can directly insert these security tokens into the header. SAML assertions are attached to Web Services Security messages using web services by placing assertion elements inside the `<wsse:Security>` header. The following example illustrates a Web Services Security message with a SAML assertion token.

```
<S:Envelope xmlns:S="...">
<S:Header>
      <wsse:Security xmlns:wsse="...">
          <saml:Assertion
                   MajorVersion="1"
                   MinorVersion="0"
                   AssertionID="SecurityToken-ef375268"
                      Issuer="elliotw1"
                      IssueInstant="2002-07-23T11:32:05.6228146-07:00"
                    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                    ...
           </saml:Assertion>
      </wsse:Security>
 </S:Header>
 <S:Body>
 ...
 </S:Body>
</S:Envelope>
```

For a complete list of the supported standards and specifications, read about web services specifications and APIs.

Binary security token

The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web Services Security implementation for WebSphere Application Server, Version 6 and later supports LTPA,, LTPA version 2, and X.509 certificate binary security tokens.

A binary security token has the following attributes that are used for interpretation:

- Value type
- Encoding type

The following example depicts an LTPA binary security token in a Web services security message header:

```
&ltwsse:BinarySecurityToken xmlns:wsst=
     "http://www.ibm.com/websphere/appserver/tokentype/5.0.2"
     EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
     ValueType="wsst:LTPA"&gt;
            MIZ6LGPt2CzXBQfio9wZTo1VotWov0NW3Za6lU5K7Li78DSnIK6iHj3hxXgrUn6p4wZI
            8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbFQJOqh5VcRvI+AKEaNmnEgEV65jUYAC9
            C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
            msu0sewsOKfl/WPsjW0bR/2g3NaVvBy18VlTFBpUbGFVGgzHRjBKAGo+ctkl80nlVLIk
            TUjt/XdYvEpOr6QoddGi4okjDGPyyoDxcvKZnReXww5UsoqlpfXwN4KG9as=
&lt/wsse:BinarySecurityToken&gt;
</wsse:Security>
</soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

The following example depicts an LTPA version 2 binary security token:

```
<wsse:BinarySecurityToken
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsst="http://www.ibm.com/websphere/appserver/tokentype"
  wsu:Id="ltpa_20"
  ValueType="wsst:LTPAv2">
    bRYI0Z59k/P1gIkgSaxeJIQoI1BdojxjdoD+6qMmiH37lqS6U90Wx6EArMA05FHVyTmxvIJACGD
    UVfqVcPDQCdPlWAn9Brhz/bXw9OEVx0wx/eNYQuiBvEVNam7urd8SxZkqppOZyeN6APZ4Z4Rox0M
    jqQv9lFIB/AKBpJyaK8V9Z9gFO8k6J5HmE/G9jdBov9Su6hXlfF50Bhy6tx8BEm4Zn/pkeNc1H1d+
    tOxwDOfS0ORWH0tjzDCTFpAMPjMmfR0/o7o3DivONtZG61ylbcwB4hx01iQC/FN5DJwrEy8kCwCeF
    ywubKVVt5pyM1k6uVXI8ik5Pjf9aU1ei86y5iXc9CirhvqosXiZvjObHTYKZSjtGiMYw3q9NKbZxs
    SzfCuAdht8sjGfaVo43i0iz7CuFYAywqVldUPjwSTvCGNtmWB/3MRtBDrmq3fqYSomjw5ZWDFex/n
    98ZaOz8mUjNHinJc4APTtEx6S10CxUkUc8b8hoCdqbcOGdZcGqYF7xgcFXvsezsXw0eRmhra54x6g
    CJs1skMMNvi0vF2pic1cg4GClQ74NKxV1oTrDZPaQPTikYGJOLKHBPYnbPda0hPkX+iCOYN0IIRBa
    Vwjj1T0G+Y/MgokiNJRGwUQ7VHXEo0+Q2HsmCkmAFrIp4lZc9fGcFyVY/EUBBpkGchL0eKNv4DoVJW
    6EhFXWZdeiVk8
</wsse:BinarySecurityToken>
```

The following example depicts an X.509 v3 binary security token containing an X.509 public certificate:

```
<wsse:BinarySecurityToken
  EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
  ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
  wsu:Id="x509bst_12"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    MIIDQDCCAqmgAwIBAgICAQUwDQYJKoZIhvcNAQEFBQAwTjELMAkGA1UEBhMCSlAxETAPBg
    NVBAgTCEthbmFnYXdhMQwwCgYDVQQKEwNJQk0xDDAKBgNVBAsTA1RSTDEQMA4GA1UEAxMH
    SW50IENBMjAeFw0wMTEwMDExMDAwMzlaFw0xMTEwMDExMDAwMzlaMFMxCzAJBgNVBAYTAk
    pQMREwDwYDVQQIEwhLYW5hZ2F3YTEMMAoGA1UEChMDSUJNMQwwCgYDVQQLEwNUUkwxFTAT
    BgNVBAMTDFNPQVBQcm92aWRlcjCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAraakNJ
    1JzkPUuvPdXRvPOOCl12nBwmqvt65dk/x+QzxxarDNwH+eWRbLyyKcrAyd0XGV+Zbvj6V3
    O9DSVCZUCJttw6bbqqeYhwAP3V8s24sID77tk3gOhUTEGYxsljX2orL26SLqFJMrvnvk2F
    RS2mrdkZEBUG97mD4QWcln4d0CAwEAAaOCASYwggEiMAkGA1UdEwQCMAAwCwYDVR0PBAQD
    AgXgMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBg
    NVHQ4EFgQUlXSsrVRfZOLGdJdjEIwTbuSTe4UwgboGA1UdIwSBsjCBr4AUvfkg1Tj5ZHLT
    29p/3M6w/tC872+hgZKkgY8wgYwxCzAJBgNVBAYTAkpQMREwDwYDVQQIEwhLYW5hZ2F3YT
    EPMA0GA1UEBxMGWWFtYXRvMQwwCgYDVQQKEwNJQk0xDDAKBgNVBAsTA1RSTDEZMBcGA1UE
    AxMQU09BUCAyLjEgVGVzdCBDQTEiMCAGCSqGSIb3DQEJARYTbWFydXlhbWFAanAuaWJtLm
    NvbYICAQEwDQYJKoZIhvcNAQEFBQADgYEAXE7mE1RPb3lYAYJFzBb3VAHvkCWa/HQtCOZd
    yniCHp3MJ9EbNTq+QpOHV60YE8u0+5SejCzFSOHOpyBgLPjWoz8JXQnjV7VcAbTglw+ZoO
    SYy64rfhRdr9giSs47F4D6woPsAd2ubg/YhMaXLTSyGxPdV3VqQsutuSgDUDoqWCA=
</wsse:BinarySecurityToken>
```

X.509 Binary Security Token

An X.509 binary security token is the base64 encoded representation of an X.509 public certificate.

The following table describes the X.509 token type.

X.509 token type	Description
X.509 version 1	Contains just the X.509 public certificate.
X.509 version 3	Contains just the X.509 public certificate.
PKIPath	Contains an ordered list of X.509 public certificates packaged in a PKIPath. The X509PKIPathv1 token type may be used to represent a certificate path.
PKCS7	Contains a list of X.509 certificates and, optionally, certificate revocation lists (CRLs) packaged in a PKCS#7 wrapper. The PKCS7 token may be used to represent a certificate path.

X.509 tokens are generally used to protect a SOAP message with XML Digital Signature or XML Encryption. Although not recommended, an X.509 token can also be used as an authentication token.

Using X.509 tokens for Authentication

When you authenticate a token, you are verifying that the sender of a token is who he says he is. You take a piece of public information that is sent in the message, such as a a user id, and verify it somehow with a piece of private information that only they can provide, such as a password.

As a very simple example, when you authenticate a UsernameToken, the user name and password are passed in the SOAP message and they are checked against the user registry at the endpoint.

For an X.509 certificate, the public information is the public key/DN and the private information is the private key. Unlike the password for a UsernameToken, the private key is not sent in the message.

When an X.509 token is used to sign a message, the following process is used:

1. If trust is enabled, the certificate is evaluated against the trust store and cert store, if configured. This will catch trust errors, certificate chaining errors, revocation errors, certificate expiration, etc. For example, you can have specific DNs in the trust store to trust each certificate explicitly, or just the root CA to trust all certificates issued from this CA, but no others.
2. The runtime verifies that the sender of the message has the private key associated with the certificate by verifying the signature. If the signature cannot be verified, then one of the following conditions occurred:
   a. The message was signed with a private key that did not match the public key in the message.
   b. The message was modified after it was sent.

After the signature is verified, you know that the sender of the message is the holder of the private key; you know that he is who he says he is.

If you pass an X.509 token in a message without using its private key to sign the message, you will not perform step #2. You will not verify that the sender of the message is the holder of the private key, or he is who he says he is. When you sign a message, you are doing something that only the holder of the private key can do.

With an unprotected X.509 token, that is, the x.509 token was not used to sign the message, the system can be compromised in the following way:

> Capture of the valid message and substitution of an attacker's X.509 token in the same message.

It is not recommend that an unprotected X.509 token be sent in a message. If an X.509 token is to be used for authentication, it is recommend that one X.509 token be sent in the message and that X.509 token be used for both Digital Signature and authentication, with a caller configuration. The signing part reference is used in the caller settings.

The following table describes the X.509 token value types.

X.509 token type	Value type
X.509 Version 1	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509
X.509 Version 3	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3
PKIPath	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

X.509 token type	Value type
PKCS7	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Kerberos token

IBM WebSphere Application Server provides Kerberos token support for web services message-level security. The support is based on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Kerberos Token Profile Version 1.1. Use this topic to understand the Kerberos support that is available for web services.

Kerberos token profile version 1.1

Kerberos Version 5 is a mature, open standard that provides a secure third-party authentication mechanism. The OASIS Web Services SOAP Message Security specification references the Kerberos token in the SOAP message. Web services applications can use the Kerberos token to send identities and protect messages more securely. Overall, Kerberos support involves Kerberos support in Java Platform, Enterprise Edition (Java EE) security and the Kerberos token support in Web Services Security. This topic covers the Kerberos token support in Web Services Security only.

In WebSphere Application Server Version 7.0 and later, Web Services Security supports the Kerberos token, which is based on OASIS WS-Security Kerberos Token Profile Version 1.1 specification. The Kerberos token is a binary security token for web services message-level security. Web Services Security provides SOAP message-level security, such as security token propagation, message signature, and message encryption. The Kerberos token is used for message security, specifically with the SOAP message security specification for web services, and is another supported token, such as the username token and the secure conversation token.

For more information, see the Web Services Security Kerberos Token Profile Version 1.1 specification. The specification explains how to use Kerberos security with the Web Services Security and how the Kerberos token is propagated and used to secure the SOAP message through signing and encryption.

Kerberos token profile enablement

The WebSphere Application Server configuration model leverages existing tools and frameworks for the Kerberos token profile configuration of authentication and message protection, such as:

- Policy set and binding configuration to enable the Kerberos token profile for Java API for XML-Based Web Services (JAX-WS) applications
- Deployment descriptor and binding configuration to enable the Kerberos token profile for JAX-RPC applications
- Token profile enablement with a Kerberos token for JAX-WS applications
- Minimal client configuration to enable the Kerberos token profile using the JAX-WS programming model

For JAX-WS client applications, the design updates the application programming interfaces (APIs) for Web Services Security and enforces a Web Services Security policy with a Kerberos token, which is based on the OASIS token profile. To enable a Kerberos token profile by using a policy set, you must first establish the Web Services Security policy and binding files by using a custom token. For more information, see the “Kerberos configuration models for web services” topic.

Kerberos support

The following Kerberos-related function is supported by web services in WebSphere Application Server:

- Client programming models for JAX-WS applications with Web Services Security APIs

- Interoperability with Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET
- Recovery of web services message security tokens for JAX-WS applications
- Kerberos token profile enablement
- Integration with the base security for the application server
- Kerberos token generation for the client and service
- Kerberos consumption at the service
- Clustering and high-availability for JAX-WS applications
- Kerberos token profile configuration of authentication and message protection for JAX-WS applications
- Integration in a single realm with either a Microsoft or z/OS operating system Key Distribution Center (KDC).
- Kerberos token profile configuration of authentication for JAX-RPC applications
-

The application server does not support the following function:

- Key name references
- Message protection using session keys for JAX-RPC applications
- Message protection using derived keys for JAX-RPC applications
- Generation of SHA1 keys for JAX-RPC applications
- Kerberos delegation is not supported when you are using JAX-RPC applications configured with the Kerberos authentication security mechanism
- A Kerberos token is not recoverable when JAX-WS applications are enabled with web services Reliable Messaging

Kerberos message protection for web services:

Message-level security is based on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Kerberos Token Profile Version 1.1 specification. Use this topic to gain an overall understanding of how message protection is implemented with a Kerberos token for web services.

Message protection

The application server can interoperate with other web services technology because of the implementation of the OASIS web services Kerberos token profile. This specification defines the standards for securing a SOAP message with the Kerberos token. However, mutual authentication is not defined by the token profile. The OASIS Web Services SOAP Message Security specification describes how to secure a SOAP message through signing and encryption by using and referencing a Kerberos token. Specifically, the OASIS specification defines how the Kerberos token, as a wrapped or unwrapped AP_REQ packet, is encoded and attached to the SOAP message. The token that is described in the OASIS Kerberos token profile is limited to the AP_REQ packet, which consists of a service ticket and an authenticator. The AP_REQ packet is obtained from the Key Distribution Center (KDC), which serves as the third-party authentication service.

Multiple formats exist for the Kerberos token, as defined in the OASIS Web Services Security Kerberos Token Profile 1.1. The `@ValueType` attribute is used to specify the token format. You must specify one of the following `<@ValueType>` attributes for the element:

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120

The resulting AP_REQ token can be either GSS-API framed (wrapped) or raw (unwrapped). The token must be Base-64 encoded.

Kerberos usage overview for web services:

You can use a Kerberos token to complete similar functions that you might currently complete with other binary security tokens, such as Lightweight Third Party Authentication (LTPA) and Secure Conversation tokens.

Token generator

After the Kerberos token is created from the Key Distribution Center (KDC), the Web Services Security generator encodes and inserts the token into the SOAP message and propagates the token for token consumption or acceptance. If a message integrity or confidentiality key is required, a Kerberos sub-key or a Kerberos session key from the Kerberos ticket is used. A key can be derived from either the Kerberos sub-key or the Kerberos session key. Web Services Security uses the key from the Kerberos token to sign and encrypt the message parts as described in the OASIS Web Services Security Kerberos Token Profile Version 1.1 specification. The type of key to use is predetermined by the Web Services Security configuration or policy. Also, the size of the derived key is configurable.

The value of the signature or encryption key is constructed from the value of one of the following keys:

- The Kerberos sub-key when it is present in the authenticator
- A session key directly from the ticket if the sub-key is absent
- A key that is derived from either of the previous keys

When the Kerberos token is referenced as a signature key, the signature algorithm must be a hashed message authentication code, which is http://www.w3.org/2000/09/xmldsig#hmac-sha1. When the Kerberos token is referenced as an encryption key, you must use one of the following symmetric encryption algorithms:

- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc
- http://www.w3.org/2001/04/xmlenc#tripledes-cbc

Attention:

- The Application Server supports Kerberos Version 5 only.
- You can use a AES-type symmetric algorithm suite in Web Services Security when the Kerberos ticket complies with RFC-4120 only.
- A Kerberos key with the RC4-HMAC 128–bit key type only is used when the KDC is on a Microsoft Windows 2003 server.
- A Kerberos key with AES 128–bit or 256–bit key types is used when the KDC is on a Microsoft Windows 2008 server.
- A Kerberos ticket must be forwardable and not contain an address when the service provider is running in a cluster.
- You must import an unrestricted Java security policy when you use an AES 256–bit encryption algorithm.

For information about using a Kerberos token in a cross or trusted realm environment, read the topic "Kerberos token security in a single, cross, or trusted realm environment."

Token consumer

The Web Services Security consumer receives and extracts the Kerberos token from the SOAP message. The consumer then accepts the Kerberos token by validating the token with its own secret key. The secret key of the service is stored in an exported keytab file. After acceptance, the Web Services Security consumer stores the associated request token information into the context Subject. You can also derive the corresponding key to the request token. The key is used to verify and decrypt the message. If the request token is forwardable and does not contain an address, the application server can use the stored token for downstream calls.

Token format and reference

For JAX-WS applications, use the existing custom policy set or administrative command scripts for the custom policy to specify the Kerberos token type, the message signing, and message encryption. The JAX-WS programming model for WebSphere Application Server provides minimal configuration to enable the Kerberos token profile with the Kerberos token.

For JAX-RPC applications, use the deployment descriptor to specify that the custom token use the Kerberos token. You can use the Kerberos token for authentication, but you cannot use it for message signing or encryption.

WebSphere Application Server supports the following callback handler classes for the Kerberos Version 5 token:

- com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler

  This class is a callback handler for Kerberos Version 5 token on the consumer side. This instance is used to generate the WSSVerification and WSSDecryption objects to validate a Kerberos binary security token.
- com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler

  This class is a callback handler for Kerberos Version 5 token on the generator side. This instance is used to generate the WSSSignature object and the WSSEncryption object to generate a Kerberos binary security token.

The OASIS Web Services Security Kerberos Token Profile Version 1.1 specification states that the Kerberos token is attached to the SOAP message with the `<wsse:BinarySecurityToken>` element. The following example shows the message format. The boldface type shows delineates the binary security token information from the other parts of the example.

```
<S11:Envelope xmlns:S11="..." xmlns:wsu="...">
   <S11:Header>
      <wsse:Security xmlns:wsse="...">
          <wsse:BinarySecurityToken
              EncodingType="http://docs.oasis-open.org/wss/2004/01/
                 oasis-200401-wss-soap-message-security-1.0#Base64Binary"
              ValueType="http://docs.oasis-open.org/wss/
                 oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ"
              wsu:Id="MyToken">boIBxDCCAcCgAwIBBaEDAgEOogcD...
          </wsse:BinarySecurityToken>
          ...
      </wsse:Security>
    </S11:Header>
    <S11:Body>
        ...
    </S11:Body>
</S11:Envelope>
```

The Kerberos token is referenced by the `<wsse:SecurityTokenReference>` element. The `<wsu:Id>` element, which is specified within the `<wsse:BinarySecurityToken>` element and is shown within the following example in boldface type, directly references the token in the `<wsse:SecurityTokenReference>` element.

The @wsse:TokenType attribute value within the <wsse:SecurityTokenReference> element matches the ValueType attribute value of the <wsse:BinarySecurityToken> element. The Reference/@ValueType attribute is not required. However, if the attribute is specified, its value must be equivalent to the @wsse11:TokenType attribute.

The following example shows the message format, the correlation between the<wsu:Id> and <wsse:SecurityTokenReference> elements, and the relationship between the @wsse:TokenType and ValueType attribute values.

```
<S11:Envelope xmlns:S11="..." xmlns:wsu="...">
   <S11:Header>
      <wsse:Security xmlns:wsse="...">
          <wsse:BinarySecurityToken
               EncodingType="http://docs.oasis-open.org/wss/2004/01/
                    oasis-200401-wss-soap-message-security-1.0#Base64Binary"
               ValueType="http://docs.oasis-open.org/wss/
                    oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ"
               wsu:Id="MyToken">boIBxDCCAcCgAwIBBaEDAgEOogcD...
          </wsse:BinarySecurityToken>
          </wsse:Security>
   </S11:Header>
</S11:Envelope>
      <wsse:Security>
       </wsse:Security>
          <wsse:SecurityTokenReference
               TokenType="http://docs.oasis-open.org/wss/
                    oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ">
               <wsse:Reference URI="#MyToken"
                    ValueType="http://docs.oasis-open.org/wss/
                        oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ">
              </wsse:Reference>
          </wsse:SecurityTokenReference>
         ...
      <wsse:Security>
      </wsse:Security>
    <S11:Header>
    </S11:Header>
    <S11:Body>
        ...
    </S11:Body>
 <S11:Envelope>
 </S11:Envelope>
```

The <wsse:KeyIdentifier> element is used to specify an identifier for the Kerberos token. The value of the identifier is a SHA1 hash value of the encoded Kerberos token in the previous message. The element must have a ValueType attribute with a #Kerberosv5APREQSHA1 value. The KeyIdentifier reference mechanism is used on subsequent message exchanges after the initial Kerberos token is accepted. The following example shows the key identifier information in boldface type:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
    <S11:Header>
      <wsse:Security>
         ...
         <wsse:SecurityTokenReference
               wsse11:TokenType=http://docs.oasis-open.org/wss/
                   oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ>
               <wsse:KeyIdentifier
                   ValueType="http://docs.oasis-open.org/wss/
                       oasis-wss-kerberos-token-profile-1.1#Kerberosv5APREQSHA1">
                       GbsDt+WmD9XlnUUWbY/nhBveW8I=
               </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
          ...
      </wsse:Security>
   </S11:Header>
   <S11:Body>
       ...
    </S11:Body>
</S11:Envelope>
```

Multiple references to the Kerberos token

The client is not required to send a Kerberos token in every request after the Kerberos identity is validated and accepted by the service. The OASIS Web Services Security Kerberos Token Profile Version 1.1 specification suggests that you use a SHA1 encoded key with the <wsse:KeyIdentifier> element within the <wsse:SecurityTokenReference> element for every subsequent message after the initial AP_REQ packet is accepted. However, the runtime environment for Web Services Security must map the key identifier to a cached Kerberos token for further processing. IBM WebSphere Application Server 7.0 and later supports this SHA1 caching as described in the profile, by default. However, the application server also provides the ability to generate new AP_REQ tokens for each request with the existing service Kerberos ticket. When you interoperate with Microsoft .NET, do not use pSHA1 caching; generate an AP_REQ packet for each request.

Kerberos configuration models for web services:

The IBM WebSphere Application Server configuration model leverages existing frameworks.

The configuration model features include:

- Deployment descriptors and bindings configuration to enable the Kerberos token profile for Java API for XML-based RPC (JAX-RPC) applications
- Policy sets and bindings configuration to enable the Kerberos token profile for Java Architecture for XML Web Services (JAX-WS) applications
- Web Services Security APIs for JAX-WS applications
- Administrative command scripts
- Interoperability with Microsoft Web Services Enhancements (WSE) Version 3.5

Following are some examples of possible configurations when using the Kerberos token:

- A JAX-WS client on Windows operating systems
- A JAX-RPC client on Windows operating systems
- A Windows JAX-RPC client on z/OS operating systems
- Web Services Security APIs on Windows operating systems
- A Microsoft .NET WSE 3.5 client on Windows operating systems
- A Microsoft .NET WSE 3.5 client on z/OS operating systems

JAX-WS configuration model

For JAX-WS applications, the WebSphere Application Server client configuration model uses the policy set and leverages a custom policy set for the Kerberos token. You can specify the Kerberos token type and message signing and the encryption by using the custom policy set. The Web Services Security (WS-Security) policy is the security policy that is used to secure the application messages.

Using the administrative console, you can specify the Kerberos token type, message signing, and message encryption by using an existing custom policy set. Kerberos token generation and consumption includes the Kerberos token generation for unmanaged JAX-WS clients.

The JAX-WS programming model also provides capabilities to enable the Kerberos token profile and identity assertion by configuring the Kerberos token using policy sets, Web Services Security APIs, and administrative command scripts.

For JAX-WS applications, you can use administrative commands to configure the policy set as an alternative to using the administrative console.

JAX-RPC configuration model

JAX-RPC applications are configured using a deployment model. The deployment descriptor specifies the custom token to use for the Kerberos token. A JAX-RPC client can generate the specified Kerberos token. A JAX-RPC web service can successfully authenticate the Kerberos token by using a custom or the default Kerberos identity mapping login module.

API configuration model

A set of APIs is provided by WebSphere Application Server. To successfully use these APIs, application developers must have knowledge about the OASIS Web Services Security Version 1.0 and 1.1 specifications. When you use these APIs, the application server assumes that a policy set is not attached to the client resources; however, a warning is still issued when the application server detects any policy set information.

For JAX-WS client applications, the APIs include and enforce Web Services Security policy for the Kerberos token, which is based on the OASIS token profile. To enable the Kerberos token profile with the policy set, you must first configure the WS-Security policy and the binding files with the custom token.

For JAX-RPC applications, APIs for Web Services Security are not provided. You must use the deployment descriptor to specify the custom token to use the Kerberos token. You can use the custom token panels within an assembly tool, such as Rational Application Developer, to configure the deployment information.

Kerberos clustering for web services:

Clusters are groups of servers that are managed together and participate in workload management.

In a clustered environment, the Kerberos token needs to be distributed and recoverable. The Web Services Security configuration saves and distributes Kerberos tokens among the cluster members. The Kerberos tokens that are created or validated in one server are available to the other cluster members. The distributed cache or database repository need to be configured as the caching mechanism.

Web Services Security Kerberos token for authentication in a single or cross Kerberos realm environment:

To secure web services messages, you can use a Kerberos token as either an authentication token or a message protection token. For Kerberos authentication, both the single Kerberos realm environment, and the cross or trusted Kerberos realm environment are supported.

Single realm environment

In a single Kerberos realm environment, both the client application and the service provider use the same Kerberos realm. The client application obtains a Kerberos token based on the Kerberos realm used by the service provider. To configure the token, the client application defines the Kerberos service principal name (SPN) for the service provider in the client policy token generator bindings. The format of the SPN is shown later in this section, where `Kerberos_Realm_Name` is optional.

```
ServiceName/HostName@Kerberos_Realm_Name
```

For cell-level configuration in WebSphere Application Server, all service providers use the same Kerberos realm.

If the service provider uses the Kerberos identity from the client for downstream web services requests, a delegated Kerberos ticket must exist in the Kerberos token that is specified in the Kerberos configuration file. The system JAAS login module for Kerberos is added to the provided Web Services Security caller. For more information on using the Kerberos token for caller credentials, read about updating the system Java Authentication and Authorization Service (JAAS) login with the Kerberos login module, and creating a Kerberos configuration file.

Cross realm environment or trusted realm environment

The following configuration procedures must be completed for the trusted realm environment:

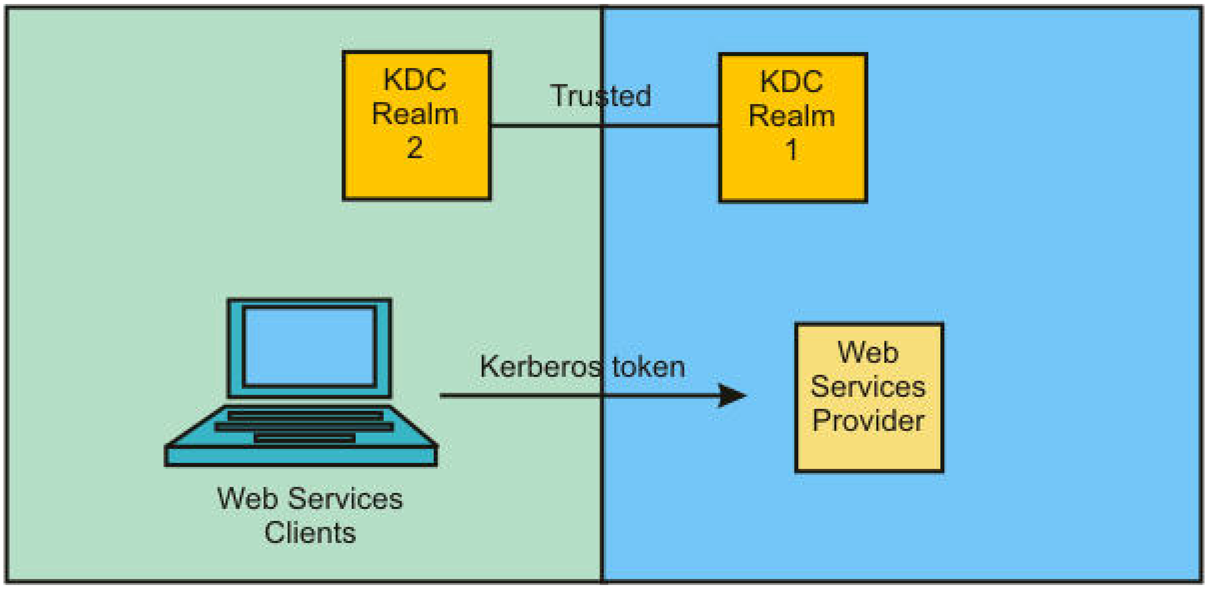
- The Kerberos trusted realm setup must be completed for all the configured Kerberos KDCs. See your Kerberos Administrator and User's Guide for more information about how to set up a Kerberos trusted realm.
- The Kerberos configuration file (krb5.ini on Windows, and krb5.conf for Unix and z/OS platforms) must list the trusted realms. See your Kerberos Administrator and User's Guide for more information.
- The client application token generator bindings must be configured with the Kerberos SPN information from the service provider. For more information, see configuring the bindings for message protection for Kerberos.

In a cross or trusted Kerberos realm environment, the client application and the service provider use different Kerberos realms that have established trust with each other. The client application obtains a Kerberos token based on the Kerberos realm used by the service provider. To configure the token, the client application defines the Kerberos SPN for the service provider in the client policy token generator bindings. The format of the SPN is shown later in this section, where `Kerberos_Realm_Name` is required.

```
ServiceName/HostName@Kerberos_Realm_Name
```

The client application must specify the Kerberos realm name for the client in the callback handler portion of the client policy token generator bindings. At the cell level, all service providers use the same Kerberos realm. However, client applications can still define their own Kerberos realm. Only peer-to-peer and transitive trust cross-realm authentication are supported.

The following figure illustrates the relationship between trusted realms as defined in the Kerberos Key Distribution Center (KDC):



If the service provider uses the Kerberos identity from the client for downstream web services requests, a delegated Kerberos ticket must exist in the Kerberos token that is configured in the Kerberos configuration file. The system JAAS login module for Kerberos is added to the provided Web Services Security caller. For more information on using the Kerberos token for caller credentials, read about updating the system JAAS login with the Kerberos login module, and creating a Kerberos configuration file.

SAML token

The Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information.

Using the product SAML function, you can apply policy sets to JAX-WS applications to use SAML assertions in web services messages and in web services usage scenarios. Use SAML assertions to represent user identity and user security attributes, and optionally, to sign and to encrypt SOAP message elements. WebSphere Application Server supports SAML assertions using the bearer subject confirmation method and the holder-of-key subject confirmation method as defined in the OASIS Web Services Security SAML Token Profile Version 1.1 specification. Policy sets and general bindings that support SAML are included with the product SAML function. To use SAML assertions, you must modify the provided sample general binding.

The SAML function also provides a set of application programming interfaces (APIs) that can be used to request SAML tokens from a Security Token Service (STS) using the WS-Trust protocol. APIs are also provided to locally generate and validate SAML tokens. For more information, read about application programming interfaces (APIs) for SAML.

Time stamp

A *time stamp* is the value of an object that indicates the system time at some critical point in the history of the object.

A time stamp is included in a message to reduce the vulnerability of an application to replay attacks. In web services, a replay attack occurs when an HTTP request is intercepted and the content is resent to the provider in its original form.

Note: When you include a time stamp in a message, you must protect its integrity using transport security, such as secure sockets layer (SSL) or message-level security, such as XML digital signature. If you do not protect the integrity of the time stamp, it is possible to capture the message and retransmit the content with a different time stamp, message expiration date, or both.

For both the JAX-RPC and JAX-WS WS-Security run times, 5 minutes is the default message expiration time that is used for the receiver if a value is not specified in the message. If a different expiration is required for a specific client or you are unsure of the target service default value, configure a message expiration time value for the outbound time stamp.

Note:

- When the Web Services Security JAX-RPC and JAX-WS run times generate or consume a message, they do not enforce that the integrity of the time stamp is protected.
- The Web Services Security JAX-RPC and JAX-WS run times do not have a default outbound message expiration value. If you want to include a message expiration value in a message, you must configure it. Although the JAX-WS run time does not have a default outbound message expiration value, you can configure an outbound message expiration value in the default general bindings. This value is acquired by all applications at the level for which the default bindings apply. For example, the value might be acquired at the cell or application level.
- For the JAX-RPC run time, the time stamp expiration value is specified in the web services deployment descriptor extension. You cannot modify the web services deployment descriptor extension from the administrative console; you can only view it. To modify the deployment descriptor extension, you must use an assembly tool and add or change the time stamp expiration value for a JAX-RPC application.
- If WS-Security constraints exist to consume a timestamp, the client must send a timestamp.

The JAX-WS WS-Security runtime complies with the OASIS WS-SecurityPolicy 1.2 specification Timestamp Required requirement. If you want to configure an application to not require an inbound time stamp when an outbound time stamp is configured you can add the com.ibm.wsspi.wssecurity.consumer.timestampRequired custom property as either an inbound or an inbound/outbound web services security custom property.

The JAX-WS runtime always puts the timestamp first, but the JAX-RPC runtime does not. If you are using the JAX-RPC WS-Security 1.0 runtime, and want to emit the Timestamp first in the Security header, you must:

- Set the property **com.ibm.wsspi.wssecurity.timestamp.keyword** to `SecurityFirst`.
- Set the property **com.ibm.wsspi.wssecurity.timestamp.dialect** to `http://www.ibm.com/websphere/webservices/wssecurity/dialect-was`. The default value for **com.ibm.wsspi.wssecurity.timestamp.dialect** is dialect-was, but for the desired function to work, the property must be set explicitly.

These properties are specified as Web Services Security property configuration settings.

Security considerations for web services

When you configure Web Services Security, you should make every effort to verify that the result is not vulnerable to a wide range of attack mechanisms. There are possible security concerns that arise when you are securing web services.

In WebSphere Application Server, when you enable integrity, confidentiality, and the associated tokens within a SOAP message, security is not guaranteed. This list of security concerns is not complete. You must conduct your own security analysis for your environment.

- Ensuring the message freshness

  Message freshness involves protecting resources from a replay attack in which a message is captured and resent. Digital signatures, by themselves, cannot prevent a replay attack because a signed message can be captured and resent. It is recommended that you allow message recipients to detect message replay attacks when messages are exchanged through an open network. You can use the following elements, which are described in the Web Services Security specifications, for this purpose:

  Timestamp

  You can use the timestamp element to keep track of messages and to detect replays of previous messages. The WS-Security 2004 specification recommends that you cache time stamps for a given period of time. As a guideline, you can use five minutes as a minimum period of time to detect replays. Messages that contain an expired timestamp are rejected.

  Nonce

  A nonce is a child element of the <UsernameToken> element in the UsernameToken profile. Because each nonce element has a unique value, recipients can detect replay attacks with relative ease.

  Important: Both the time stamp and nonce element must be signed. Otherwise, these elements can be altered easily and, therefore, cannot prevent replay attacks.

- Using XML digital signature and XML encryption properly to avoid a potential security hole

  The Web Services Security 2004 specification defines how to use XML digital signature and XML encryption in SOAP headers. Therefore, users must understand XML digital signature and XML encryption in the context of other security mechanisms and their possible threats to an entity. For XML digital signature, you must be aware of all of the security implications resulting from the use of digital signatures in general and XML digital signature in particular. When you build trust into an application based on a digital signature, you must incorporate other technologies such as certification trust validation based upon the Public Key Infrastructure (PKI). For XML encryption, the combination of digital signing and encryption over a common data item might introduce some cryptographic vulnerabilities. For example, when you encrypt digitally signed data, you might leave the digital signature in plain text and leave your message vulnerable to plain text guessing attacks. As a general practice, when data is encrypted, encrypt any digest or signature over the data. For more information, see http://www.w3.org/TR/xmlenc-core/#sec-Sign-with-Encrypt.

- Protecting the integrity of security tokens

  The possibility of a token substitution attack exists. In this scenario, a digital signature is verified with a key that is often derived from a security token and is included in a message. If the token is substituted, a recipient might accept the message based on the substituted key, which might not be what you expect. One possible solution to this problem is to sign the security token (or the unique identifying data from which the signing key is derived) together with the signed data. In some situations, the token that is issued by a trusted authority is signed. In this case, there might not be an integrity issue. However, because application semantics and the environment might change over time, the best practice is to prevent this attack. You must assess the risk assessment based upon the deployed environment.

- Verifying the certificate to leverage the certificate path verification and the certificate revocation list

  It is recommended that you verify that the authenticity or validity of the token identity that is used for digital signature is properly trusted. Especially for an X.509 token, this issue involves verifying the certificate path and using a certificate revocation list (CRL). In the Web Services Security implementation in WebSphere Application Server Version 6 and later, the certificate is verified by the <TokenConsumer> element. WebSphere Application Server provides a default implementation for the X.509 certificate that uses the Java CertPath library to verify and validate the certificate. In the implementation, there is no explicit concept of a CRL. Rather, proper root certificates and intermediate certificates are prepared in files only. For a sophisticated solution, you might develop your own TokenConsumer implementation that performs certificate and CRL verification using the online CRL database or the Online Certificate Status Protocol (OCSP).
- Protecting the username token with a password

  It is recommended that you do not send a password in a username token to a downstream server without protection. You can use transport-level security such as SSL (for example, HTTPS) or use XML encryption within Web Services Security to protect the password. The preferred method of protection depends upon your environment. However, you might be able to send a password to a downstream server as plain text in some special environments where you are positive that you are not vulnerable to an attack.

Securing web services involves more work than just enabling XML digital signature and XML encryption. To properly secure a Web service, you must have knowledge about the PKI. The amount of security that you need depends upon the deployed environment and the usage patterns. However, there are some basic rules and best practices for securing web services. It is recommended that you read some books on PKI and also read information on the Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP).

Nonce, a randomly generated token:

Nonce is a randomly-generated, cryptographic token that is used to prevent replay attacks. Although nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

Without nonce, when a UsernameToken is passed from one machine to another machine using a nonsecure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same password might be reused when the user name token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption. However, nonce alone, used in a non-secure transport, cannot adequately address the replay problem. Nonce is most useful when the SOAP message is transmitted via a communication channel that is secured, either at the transport level, or at the message level.

To help eliminate these replay attacks, the <wsse:Nonce> and <wsu:Created> elements are generated within the <wsse:UsernameToken> element and used to validate the message. The server checks the freshness of the message by verifying that the difference between the nonce creation time, which is specified by the <wsu:Created> element, and the current time falls within a specified time period. Also, the server checks a cache of used nonces to verify that the user name token in the received SOAP message has not been processed within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

To add a nonce for the UsernameToken, you can specify it in the token generator for the user name token. When the token generator for the UsernameToken is specified, you can select the **Add nonce** option if you want to include nonce in the user name token.

Basic Security Profile compliance tips:

The Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP) 1.0 promotes interoperability by providing clarifications and amplifications to a set of nonproprietary web services specifications. WebSphere Application Server Web Services Security provides configuration options to ensure that the BSP recommendations and security considerations can be enabled to ensure interoperability. The degree to which you follow these recommendations is then a measure of how well the application you are configuring complies with the Basic Security Profile (BSP).

Support for applications to comply to the Basic Security Profile (BSP) is new in WebSphere Application Server Version 8.5. For more information on the Basic Security Profile, see Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP), Basic Security Profile Version 1.0.

You can use either a predefined list of keywords or XPath expressions to comply to the BSP. Both the keywords and the XPath expressions are specified in the deployment descriptor configuration file and are configured using an assembly tool.

Basic Security Profile recommendations

Follow these recommendations to ensure that your configured applications are Basic Security Profile (BSP) compliant.

- Do not use the original XPath transform, http://www.w3.org/TR/1999/REC-xpath-19991116

  When you refer to an element in a SECURE_ENVELOPE that does not carry an ID attribute type from a ds:Reference in a SIGNATURE element, you must use the XPath Filter 2.0 transform, http://www.w3.org/2002/06/xmldsig-filter2 to refer to that element.

  Any ds:Transform/@Algorithm attribute in a SIGNATURE element must have one of these values:

  – http://www.w3.org/2001/10/xml-exc-c14n#

  – http://www.w3.org/2002/06/xmldsig-filter2

  – `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform`

  – http://www.w3.org/2000/09/xmldsig#enveloped-signature

  – `http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform`

  – `http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform`
- Do not use the http://www.w3.org/2000/09/xmldsig#dsa-sha1 signature algorithm.

  Any ds:SignatureMethod/@Algorithm element in a SIGNATURE that is based on a symmetric key must have one of the following values:

  – http://www.w3.org/2000/09/xmldsig#rsa-sha1

  – http://www.w3.org/2000/09/xmldsig#hmac-sha1
- Do not specify the digestvalue keyword for the message part to encrypt. Instead, use the signature keyword.

  If the value of a ds:DigestValue element in a SIGNATURE element requires encryption, the entire parent ds:Signature element must be encrypted. A SIGNATURE must not have any xenc:EncryptedData elements among its descendants.
- Do not use the KEYNAME key information type

  KEYNAME references can be ambiguous and compliance with the BSP disallows the use of KEYNAME.

  A SECURITY_TOKEN_REFERENCE must not use a key name to reference a SECURITY_TOKEN. The child element of a ds:KeyInfo element in an ENCRYPTED_KEY must be either a SECURITY_TOKEN_REFERENCE or a ds:MgmtData element. Using a KEYNAME key information type for an encryption key results in a KeyName child element of a ds:KeyInfo element and is disallowed for BSP compliance.
- Do not use the http://www.w3.org/2001/04/xmlenc#aes192-cbc bit data encryption algorithm.

Any xenc:EncryptionMethod/@Algorithm attribute in an ENCRYPTED_DATA element must have one of these values:

  - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - http://www.w3.org/2001/04/xmlenc#aes128-cbc
  - http://www.w3.org/2001/04/xmlenc#aes256-cbc

- Do not use the advanced encryption standard (AES) key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192 key encryption algorithm.

  When used for key wrap, any xenc:EncryptionMethod/@Algorithm attribute in an ENCRYPTED_KEY element must have one of these values:

  - http://www.w3.org/2001/04/xmlenc#kw-tripledes
  - http://www.w3.org/2001/04/xmlenc#kw-aes128
  - http://www.w3.org/2001/04/xmlenc#kw-aes256

Configuration Options for BSP Compliance

You achieve BSP compliance when certain configuration choices are made. The assembly tool assists you in using appropriate choices when configuring the application by issuing warning messages. The following configuration descriptions comprise these warnings:

- When configuring the ds:Transforms element in a signature, the list of transforms must include as its last child element http://www.w3.org/2001/10/xml-exc-c14n# or `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform`
- Add a wsse:Nonce or wsse:Created element to a Username token to prevent replay. After the element is added, sign the Username token to prevent undetected alteration of these fields; otherwise, replay can occur.

Distributed nonce cache:

In previous releases of WebSphere Application Server, the nonce was cached locally. WebSphere Application Server Versions 6 and later use distributed nonce caching. The distributed nonce cache makes it possible to replicate nonce data among servers in a WebSphere Application Server cluster.

If nonce elements are in a SOAP header, all nonce values are cached by the server in the cluster. If the distributed nonce cache is enabled, the cached nonce values are copied to other servers in the same cluster. Then, if the message with the same nonce value is sent to (one of) other servers, the message is rejected. A received nonce cache value is cached and replicated in a push manner among other servers in the cluster with the same replication domain. The replication is an out-of-process call and, in some cases, is a remote call. Therefore, there is latency when the content of the cache in the cluster is updated.

For example, you might have application server A and application server B in cluster C.

- A SOAP client sends a message with nonce `abc` to application server A.
- The server caches the value and pushes it to the other application server B.
- If the client sends the message with nonce `abc` to application server B after a certain time frame, the message is rejected and if the application server B receives the nonce with the same value within a specified period of time, a SoapSecurityException is thrown by application server B.

  For more information, see the information that explains nonce cache timeout, nonce maximum age, and nonce clock skew in Token generator configuration settings.
- If the client sends the message with another nonce value of `xyz`, the message is accepted, the value is cached by application server B and is copied into the other application servers within the same cluster.

Web Services Security token propagation

Web Services Security has the ability to send security tokens in the security header of a SOAP message. These security tokens can be used to sign, verify, encrypt or decrypt message parts. Security tokens can also be sent as stand-alone security tokens and set as the caller on the request consumer. *Web Services Security token propagation* is used to send these stand-alone security tokens in a wsse:BinarySecurityToken element within the security header of the SOAP message.

Web Services Security has the following built-in token types:

- Username token
- X.509 token
- Lightweight Third-Party Authentication (LTPA) token

You can configure Web Services Security to use custom security tokens. Web Services Security uses the same propagation token format as the Security attribute propagation feature. Web Services Security can propagate all of the built-in security token types and can propagate custom token types as long as they are serializable by the security attribute propagation feature.

When you configure a propagation token in a token generator or token consumer, use the following values for the token type Uniform Resource Identifier (URI) and local name:

- **Token type URI:** http://www.ibm.com/websphere/appserver/tokentype
- **Token type local name:** LTPA_PROPAGATION

When a propagation token is generated, Web Services Security gathers all of the serializable security tokens in the *RunAs* subject for the current thread and serialize the security tokens within a wsse:BinarySecurityToken token. To have a RunAs subject and the credentials that are necessary on the current thread, a JAAS login must occur on the current thread before a propagation token can be created.

Under ordinary circumstances, for a service provider, the Java Authentication and Authorization Service (JAAS) login is achieved by including a defined caller part for the inbound token in the WS-Security configuration. For a web services client, the JAAS login is achieved by configuring HTTP basic authentication.

There are two common uses for a propagation token:

- A client from within a secured service propagates the serializable security tokens and credentials from the current RunAs subject to a downstream server.
- A server-based client that is secured in the web container with HTTP basic authentication can use a propagation token.

  For a server-based client, the overhead for propagation tokens is not necessary as only the identity is required and not the full set of credentials. However, if the client application makes modifications to the subject after it is invoked by the web container, it might be appropriate to use a propagation token. If only an identity token is required, an ordinary LTPA token might be appropriate. You can generate this LTPA token from the RunAs subject that is created by the JAAS login.

Important: For the receiver of the LTPA propagation token to make proper use of the credentials that were sent to it in the propagation token, you must configure and define a caller part for the token in the WS-Security configuration on the receiver side.

Chapter 27. XML applications

This page provides a starting point for finding information about XML applications.

Overview of XML support

You can use the XML support provided with this product to work with web applications that process data using standard XML technologies like Extensible Stylesheet Language Transformations (XSLT), XML Path Language (XPath), and XML Query Language (XQuery).

XML-structured data has become the predominant format for data interchange. XML data is navigated, queried, or transformed in almost every existing WebSphere application.

Since first being standardized, XML usage in application-development environments has grown significantly to include many scenarios. WebSphere Application Server is a leading platform for the latest application development standards, including XML.

Note: IBM WebSphere Application Server Version 8.5 delivers critical technology that provides application developers with support for the following key World Wide Web Consortium (W3C) XML standards:

- Extensible Stylesheet Language Transformations (XSLT) 2.0
- XML Path Language (XPath) 2.0
- XML Query Language (XQuery) 1.0

These new and updated W3C XML standards offer application developers numerous advanced capabilities for building XML applications. Specific benefits delivered in the XPath 2.0, XSLT 2.0, and XQuery 1.0 standards include the following:

- Simpler XML application development and improved developer productivity
- Improved ability to query large amounts of data stored in XML outside of a database with XQuery 1.0
- Improved XML-application performance through new features introduced in the W3C specifications to address previous shortcomings
- Improved XML-application reliability with new support for XML schema-aware processing and validation

Note: If you want to use XPath 1.0 or XSLT 1.0 (not in backwards-compatibility mode), continue to use Java API for XML Processing (JAXP) in Java 2 Platform, Standard Edition (J2SE) 6.0 and 7.0.

For more information about these W3C XML standards, go to W3C XQuery 1.0 and XSLT 2.0 Become Standards: Tools to Query, Transform, and Access XML and Relational Data.

The product provides the IBM XML Application Programming Interface in support of these standards. This application programming interface invokes a runtime engine that is capable of executing XPath 2.0, XSLT 2.0, and XQuery 1.0 as well as manipulating the returned XML data.

The product also includes the IBM Thin Client for XML with WebSphere Application Server. The thin client allows access to the same XML API and runtime functionality (XPath 2.0, XSLT 2.0, and XQuery 1.0) available in the full product. The thin client can be copied to multiple clients running Java SE in support of a WebSphere Application Server Version 8.5 installation.

XSLT 2.0, XPath 2.0, and XQuery 1.0 major new functions

Valuable features have been added to XPath 2.0, XSLT 2.0, and XQuery 1.0 reflecting productivity and feature improvements beyond the XPath 1.0 and XSLT 1.0 standards.

XPath 2.0

- XPath 2.0 has been improved to support the XPath 2.0 and XQuery 1.0 Data Model (XDM), which is based on sequences of heterogeneous items including nodes and primitive types. This replaces and improves on the XPath 1.0 node-set support and becomes the foundation of XSLT 2.0 and XQuery 1.0 data navigation.
- XPath 2.0 adds an extensive collection of functions and operators to allow for an easier programming experience, replacing the XPath 1.0 requirement for proprietary extension mechanisms. These functions and operators help with date and time handling, enhance the string manipulation, support regular expression matching and tokenization, extend the number handling, and add functions for sequence manipulation.
- XPath 2.0 supports schema-aware processing, which allows for data navigation based on XML schema information for not only built-in schema types, but also user-defined schema types.
- XPath 2.0 adds condition (if/then/else branches), iterative (for loops), and quantified expressions (some and every tests) typical of other languages.
- XPath 2.0 adds named collations across multiple functions allowing for locale-specific operation.
- XPath 2.0 provides a backwards-compatibility mode to run most XPath 1.0 expressions unchanged.

XSLT 2.0

- XSLT 2.0 is based on XPath 2.0, allowing XSLT 2.0 to take advantage of all new XPath 2.0 features. Temporary trees have been added to allow navigation of constructed trees during transformation. User-defined functions can be defined in the XSLT language and are callable using XPath 2.0.
- XSLT 2.0 can write to multiple result documents in a single stylesheet execution.
- XSLT 2.0 supports regular expressions to analyze and separate strings.
- XSLT 2.0 allows variables and parameters to be typed, therefore improving the reliability of stylesheets and functions.
- XSLT 2.0 supports schema-aware processing, which allows XSLT 2.0 to check for valid input, temporary trees, and output documents.
- XSLT 2.0 supports initial named templates, which allows the processor to start with a defined template instead of having to match the input document, a feature commonly used with loading documents programmatically using the XPath 2.0 collection and document functions.
- Comparisons in sorting, grouping, and keys are supported with any data type and can use locale-specific named collations.
- XHTML has been added to XSLT 2.0 as a valid output format.
- The next-match instruction allows the same node to be processed with multiple templates.
- The character-map instruction allows fine grained control of serialization of characters.
- XSLT 2.0 added addition instructions for transforming and formatting dates and times.
- XSLT 2.0 added support for tunnel parameters, which allows parameters to be passed through multiple template calls without having to declare the parameter in each template call.
- XSLT 2.0 added multiple mode support to allow templates to apply to specific modes of processing within a stylesheet.
- Unparsed text can be incorporated into the data processed by a stylesheet, which then can be tokenized with the new regular expression support.
- XSLT 2.0 provides a backwards-compatibility mode to run most XSLT 1.0 stylesheets unchanged.

XQuery 1.0

- XQuery 1.0 is based on XPath 2.0, allowing XQuery 1.0 to take advantage of all new XPath 2.0 features. XQuery 1.0 builds on XPath 2.0 to provide full XML Query capability.

- XQuery's FLOWR (For, Let, Order by, Where, Return) expression allows for complicated joins across XML datasets. FLOWR allows for query of large documents or collections of documents. XQuery allows for the mixture of direct XML construction along with computed content returned from FLOWR expressions.
- XQuery has the ability to define functions and variables with syntax that is familiar to users of other languages, allowing larger programs to be defined around the data-query operations.
- XQuery 1.0 supports schema-aware processing, which allows input and constructed documents and elements to be validated.
- XQuery module support allows queries to be broken up into reusable fragments.

Overview of the XML Samples application

The XML Samples application is written to be used with the XML specifications and other documents. However, the most important function that these samples provide is a place to begin experimenting with the XML API and the supported specifications.

Limitations

The XML Samples application is not intended for deployment to production servers. It is for development and educational purposes only. All source code is provided as is for you to use, copy, and modify without royalty payment when you develop applications that run with WebSphere software. You can use the sample code either for your own internal use, for redistribution as part of an application, or in your products.

Content

- The simple API invocation examples included in the samples are intended as simple examples of using the major new features of XPath 2.0, XSLT 2.0, and XQuery 1.0.
  - XPath 2.0 examples
    - Sample 1: Simple XPath invocation

      Shows how to invoke XPath
    - Sample 2: Invoking XPath 1.0 under an XPath 2.0 run time in backwards compatibility mode

      Shows an example that demonstrates differences between XPath 1.0 and XPath 2.0 as well as how to run existing XPath 1.0 statements under XPath 2.0 in backwards-compatibility mode
    - Sample 3: Invoking schema aware XPath 2.0 expressions

      Shows how to run schema-aware expressions; shows how to load schema documents, how to validate input documents, and how to declare namespace prefixes
    - Sample 4: XPath 2.0 - document function (relative URIs) with input and output documents

      Shows how to invoke XPath using the document function with relative URIs
    - Sample 5: XPath running in compiled mode

      Shows how to invoke XPath in compiled mode
    - Sample 6: XPath running in pre-compiled mode

      Shows how to invoke XPath in pre-compiled mode
    - Sample 7: XPath 2.0 collation support

      Shows how to invoke XPath with collation support
  - XSLT 2.0 examples
    - Sample 1: Simple XSLT invocation

      Shows how to invoke XSLT
    - Sample 2: Invoking XSLT 1.0 under an XSLT 2.0 run time in backwards compatibility mode

      Shows differences between XPath 1.0 and XPath 2.0 and how to run existing XSLT 1.0 stylesheets under a XSLT 2.0 processor in backwards-compatibility mode

- Sample 3: XSLT 2.0 updated for-each support

  Shows how to use the XSLT 2.0 for-each functionality
- Sample 4: XSLT 2.0 grouping support

  Shows how to use the capability offered by xsl:for-each-group
- Sample 5: XSLT 2.0 regular expression support

  Shows how to use XSLT 2.0 regular-expression support to work with data in structured legacy formats within XML strings
- Sample 6: XSLT 2.0 date formatting

  Shows how to use XSLT 2.0 date formatting with internationalization
- Sample 7: XSLT 2.0 multiple results

  Shows how to use an XSLT 2.0 result-document instruction to write to multiple outputs simultaneously
- Sample 8: XSLT 2.0 tunnel parameters

  Shows how to use XSLT 2.0 tunnel parameters to allow values to be set and accessible during stylesheet processing
- Sample 9: XSLT 2.0 stylesheet functions

  Shows how to use the XSLT 2.0 stylesheet functions
- Sample 10: XSLT 2.0 initial template

  Shows how to use the XSLT 2.0 initial-template functionality
- Sample 11: XSLT 2.0 template with multiple modes

  Shows how to use the XSLT 2.0 template with multiple modes functionality
- Sample 12: XSLT 2.0 XHTML support - no output method specified

  Shows how to use XSLT 2.0 XHTML support with the XHTML output method
- Sample 13: XSLT 2.0 XHTML support - output method specified

  Shows how to use XSLT 2.0 XHTML support with the XHTML output method
- Sample 14: XSLT 2.0 character maps

  Shows how to use XSLT 2.0 character maps functionality
- Sample 15: XSLT 2.0 "as" attribute

  Shows how to use the XSLT 2.0 "as" attribute functionality
- Sample 16: XSLT 2.0 embedded stylesheets

  Shows how to use the XSLT 2.0 embedded stylesheets functionality
- Sample 17: XSLT 2.0 running in compiled mode

  Shows how to run XSLT in compiled mode
- Sample 18: XSLT 2.0 running in pre-compiled mode

  Shows how to run XSLT in pre-compiled mode
- Sample 19: XSLT 2.0 undeclare-prefixes serialization parameter

  Shows how to use the XSLT undeclare-prefix parameter when producing XML output that is Version 1.1 or higher
- Sample 20: XSLT 2.0 next-match

  Shows how to use the XSLT next-match functionality
- Sample 21: XSLT 2.0 usage of XPath 2.0 collection function

  Shows how to use the collection function
- Sample 22: XSLT 2.0 schema awareness - input validation (valid)

  Shows how to use the stylesheets and schemas to validate input documents
- Sample 23: XSLT 2.0 schema awareness - input validation (invalid)

  Shows how to use the stylesheets and schemas to validate input documents

- Sample 24: XSLT 2.0 schema awareness - temporary tree (valid)
  Shows how to use the validation attribute to validate temporary trees
- Sample 25: XSLT 2.0 schema awareness - temporary tree (invalid)
  Shows how to use the validation attribute to validate temporary trees
- Sample 26: XSLT 2.0 schema awareness - output document (valid)
  Shows how to use the validation attribute to validate the main output document
- Sample 27: XSLT 2.0 schema awareness - output document (invalid)
  Shows how to use the validation attribute to validate the main output document
- Sample 28: XSLT 2.0 schema awareness - element(*, T) function
  Shows how to use the stylesheets and schemas to match on element types instead of names
- Sample 29: XSLT 2.0 use-when
  Shows how to use the use-when functionality
- Sample 30: XSLT 2.0 collation support
  Shows how to use the for-each-group functionality with collations
- Sample 31: Using stylesheet-declared external functions
  Shows how to declare external functions within a stylesheet

– XQuery 1.0 examples
  - Sample 1: Simple XQuery invocation
    Shows how to invoke simple XQuery FLOWR expressions
  - Sample 2: XQuery FLWOR support - using doc function and cross document joins
    Shows how to invoke an XQuery that joins data from multiple documents
  - Sample 3: XQuery declare functions and variables
    Shows how to define and use XQuery functions and variables
  - Sample 4: XQuery TypeDeclaration support
    Shows how to use the TypeDeclaration functionality
  - Sample 5: XQuery running in compiled mode
    Shows how to run XQuery functions in compiled mode
  - Sample 6: XQuery running in pre-compiled mode
    Shows how to invoke XQuery in pre-compiled mode
  - Sample 7: XQuery operations on types (typeswitch, cast as)
    Shows how to use operations on types
  - Sample 8: XQuery schema awareness - input validation (valid)
    Shows how to validate the input document passed to the query
  - Sample 9: XQuery schema awareness - input validation (invalid)
    Shows how to validate the input document passed to the query
  - Sample 10: XQuery schema awareness - node validation (valid)
    Shows how to validate an element using the validate expression
  - Sample 11: XQuery schema awareness - node validation (invalid)
    Shows how to validate an element using the validate expression
  - Sample 12: XQuery schema awareness - element(*, T) function
    Shows how to use schema awareness to match on element types instead of names
  - Sample 13: XQuery modules support
    Shows how commonly used functions and variables can be put in a reusable library module
  - Sample 14: XQuery modules support with schema
    Shows how modules interact with schema support

  - Sample 15: Using query-declared external functions

    Shows how to declare external functions within a query

- The Blog Comment Checker examples show how you can search all or your Blogger™ web publishing service blogs for questionable comments. They are examples of high-level applications that use XPath 2.0, XSLT 2.0, and XQuery 1.0.
  - XPath Blog Checker
  - XSLT Blog Checker
  - XQuery Blog Checker
  - Database Integration Checker

Building and running a sample XML application

You can use the IBM WebSphere Application Server XML thin client, the `com.ibm.xml.thinclient_8.5.0.jar` file, to build a sample XML application. You can also use the API documentation to improve your understanding of the XML API.

Before you begin

1. Install the product.
2. Locate the `com.ibm.xml.thinclient_8.5.0.jar` file.

   You can find the `com.ibm.xml.thinclient_8.5.0.jar` file in your installation tree; for example:

To see how to build and use an application, refer to the sample application that is packaged with the product.

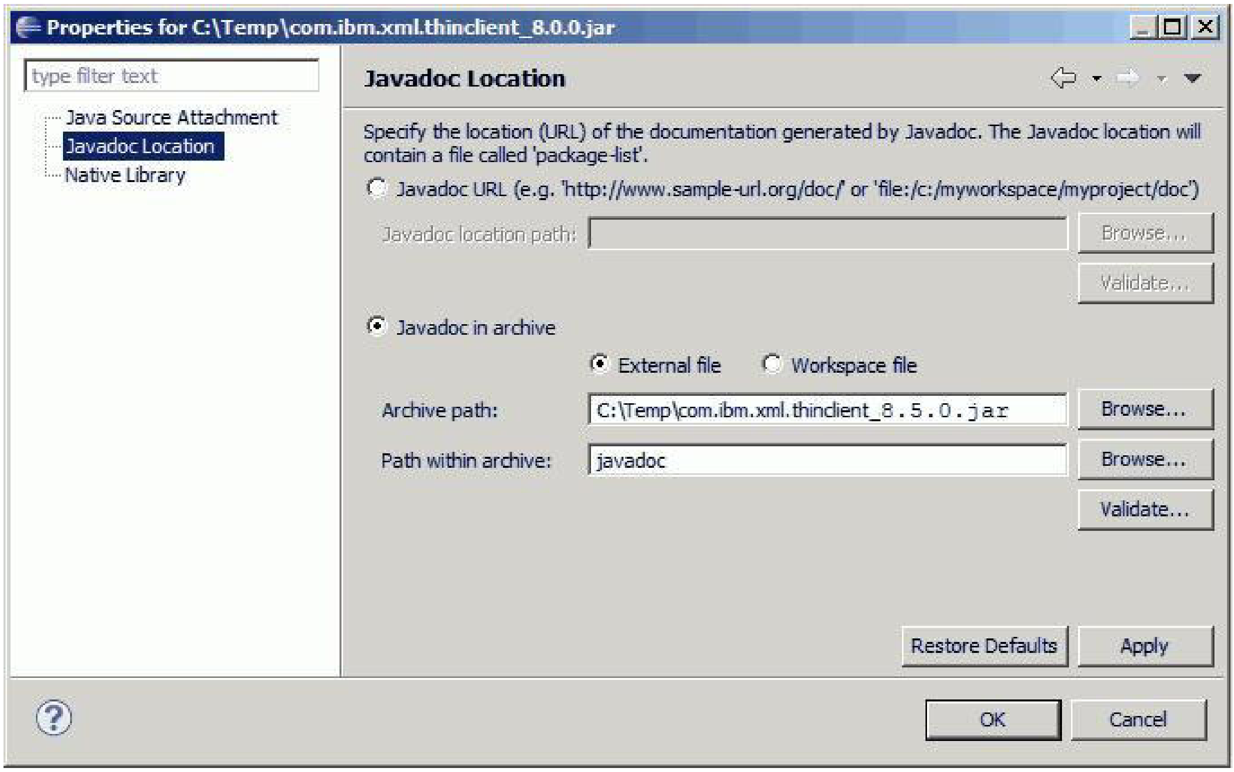
About this task

Follow this procedure when you build and run a sample XML application.

Procedure

1. For build time, include the `com.ibm.xml.thinclient_8.5.0.jar` file in the build-time class path while developing your sample XML application.

   Also, attach the API documentation from the `javadoc` directory that is inside the `com.ibm.xml.thinclient_8.5.0.jar` file.



Properties for C:\Temp\com.ibm.xml.thinclient_8.0.0.jar

type filter text

- Java Source Attachment
- Javadoc Location
- Native Library

Javadoc Location

Specify the location (URL) of the documentation generated by Javadoc. The Javadoc location will contain a file called 'package-list'.

○ Javadoc URL (e.g. 'http://www.sample-url.org/doc/' or 'file:/c:/myworkspace/myproject/doc')

Javadoc location path:

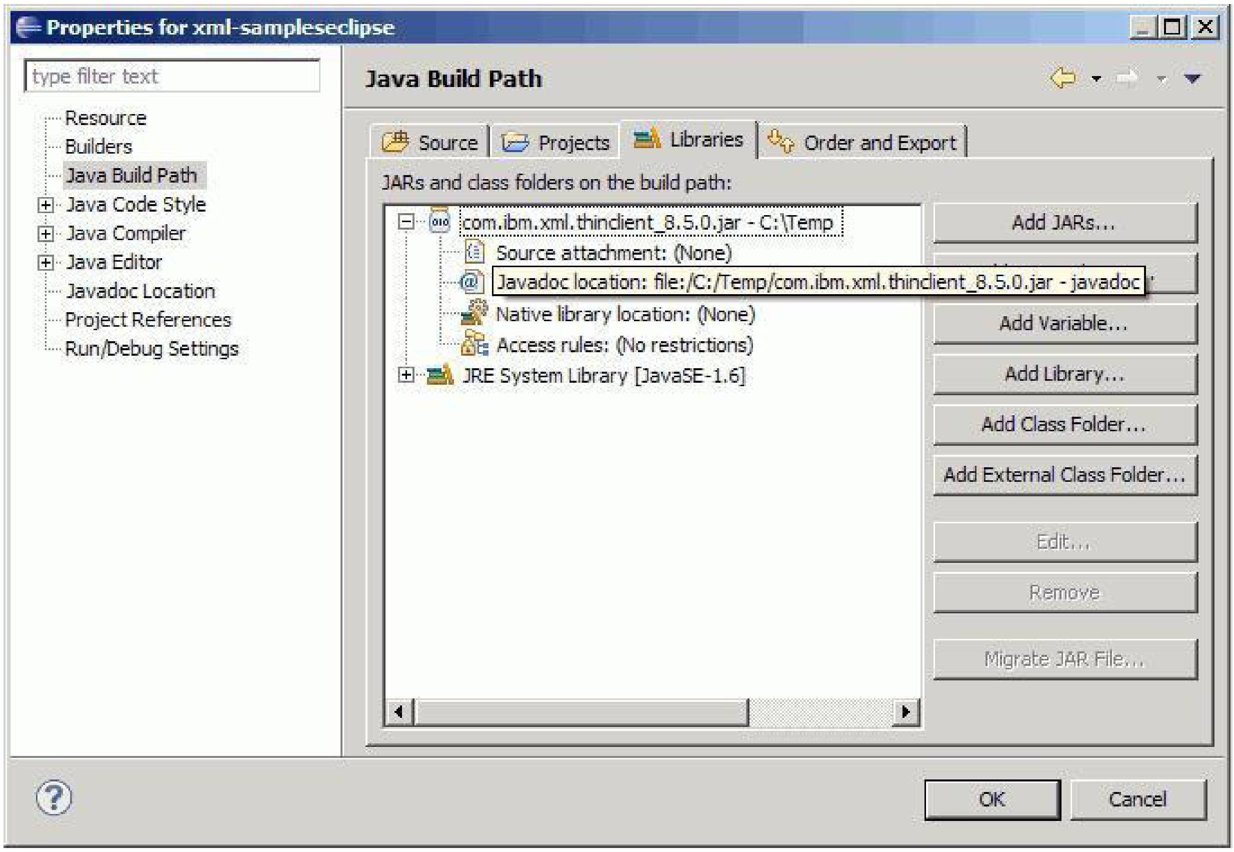
● Javadoc in archive

● External file ○ Workspace file

Archive path: C:\Temp\com.ibm.xml.thinclient_8.5.0.jar

Path within archive: javadoc

The results of these actions are shown in the following image:



Properties for xml-sampleseclipse

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Run/Debug Settings

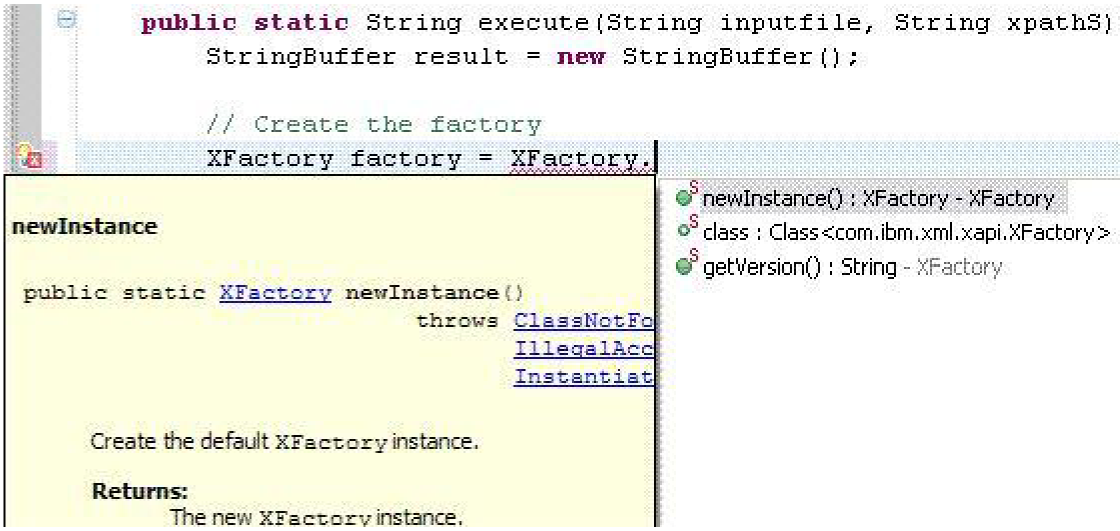
Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- com.ibm.xml.thinclient_8.5.0.jar - C:\Temp
  - Source attachment: (None)
  - Javadoc location: file:/C:/Temp/com.ibm.xml.thinclient_8.5.0.jar - javadoc
  - Native library location: (None)
  - Access rules: (No restrictions)
- JRE System Library [JavaSE-1.6]

When complete, your application should compile; and when using context completion, you should have access to the API documentation as shown here:



```
public static String execute(String inputfile, String xpathS)
    StringBuffer result = new StringBuffer();

    // Create the factory
    XFactory factory = XFactory.
```

newInstance() : XFactory - XFactory
class : Class<com.ibm.xml.xapi.XFactory>
getVersion() : String - XFactory

newInstance

```
public static XFactory newInstance()
                         throws ClassNotFo
                                IllegalAcc
                                Instantiat
```

Create the default `XFactory` instance.

Returns:
The new `XFactory` instance.

2. Deploy your application.

Chapter 28. What is new in this release

WebSphere Application Server is a proven, high-performance transaction engine that can help you build, run, integrate, and manage dynamic business applications. This release introduces key enhancements to help you succeed in the current dynamic and competitive marketplace by introducing the new lightweight Liberty profile and by offering breakthrough advances in developer productivity, application server resiliency, and improved operational efficiency.

This product excels as the foundation for a service-oriented architecture with the following main benefits:

- **"New Liberty profile: a lightweight application foundation" on page 982**

  The Liberty profile is a highly composable, fast to start, ultra lightweight profile of the application server that is optimized for developer productivity and smaller, simpler production server deployments.

- **"Fast, flexible, and simplified application development" on page 983**

  Use the new Liberty profile to jump-start application development, and speed project completion. You can download the Liberty profile from the WASdev community page, or by using the Installation Manager.

  WebSphere Application Server Version 8.5 extends its rich application development and deployment environment to also provide the following enhanced capabilities:

  – Enterprise JavaBeans (EJB) support in OSGi applications

  – Support for WebSphere SDK Java Technology Edition V7.0 as an optional pluggable JDK

  – Support for Web 2.0 and Mobile Toolkit to extend enterprise applications to mobile devices

  – Support for a new migration toolkit to help you not only migrate your applications, but to identify and resolve application problems before migration

- **Enhanced resiliency**

  Take advantage of enhanced resiliency of the application server to automatically detect and respond to certain failure scenarios. Improvements include automatic detection of certain types of memory leak and improved fault tolerance in service integration bus resiliency to handle the failures of the messaging infrastructure.

- **"Integrated tools" on page 984**

  WebSphere Application Server now provides integrated tooling to help you easily develop, assemble, and deploy your applications to WebSphere Application Server Version 8.5 and includes a rapid-deployment feature for testing applications in the Version 8.5 runtime environment.

- **"Improved operations, security, control, and integration" on page 986**

  Improve operational efficiency and controls for managing complex environments in a way that is less time consuming and resource consuming. Take advantage of the benefits of the option to select and switch between IBM WebSphere SDK Java Technology Edition Version 7.0 or 6.0 as well as powerful new batch capabilities. Enhancements for problem determination include cross component tracing (XCT), improved log and trace filtering, and support for IBM Support Assistant Data Collector. You can achieve richer application security with greater flexibility and more granular control using enhanced security with administrative audit features.

Note: The new features that are described in this topic are current as of the original release of the product version. Changes in this documentation that are related to service releases are marked with fix pack icons.

IBM i This version offers robust improvements, whether you are new to the product or making the transition from a prior release. Use this topic to obtain a high-level summary of the new features in this release. To learn more about the new and changed features in key areas that affect your specific roles in your business, see the information about what is new for installers on IBM i operating systems, administrators, security specialists, developers, and troubleshooters.

New Liberty profile: a lightweight application foundation

WebSphere Application Server includes the Liberty profile that is optimized for both operational and developer productivity. Key operational benefits include, but are not limited to, the following capabilities:

- Ultra lightweight modular runtime environment, with an install size of under 50 MB
- Incredibly fast startup time
- Java EE and OSGi application deployment support for web applications
- LDAP registry support
- Deployment, as a package, of an application and configured server
- Managed, centralized deployment to many nodes of a packaged application and server
- Platform support for distributed platforms, z/OS, and Mac OS

The Liberty profile currently supports a subset of the following parts of the full WebSphere Application Server programming model:

- Web applications
- OSGi applications
- Java Persistence API (JPA)

Familiar WebSphere Application Server enterprise qualities of service, such as security and transaction integrity, are enabled as required.

- Deploying applications to a Liberty profile server
- Network deployment of Liberty profile servers

Deploying applications to a Liberty profile server

This profile supports two models of application deployment:

- Deploy an application by dropping it into a “dropins” directory.
- Deploy an application by adding it to the server configuration.

Because it is composable, each Liberty profile server is usually configured to include only the capabilities that are needed by the applications that it serves. This approach keeps each deployed server small and agile.

Network deployment of Liberty profile servers

The job manager provides the ability to distribute and deploy a Liberty profile server and applications, and start or stop embedded resources. Administration through the job manager provides the following key features:

- Support for central administration through job manager host target jobs. You can submit job manager jobs that support the full lifecycle of Liberty profile resource deployment from initial install, to updates, to uninstall. A deployment manager is not required, although you can use the job manager function available on a deployment manager to administer the Liberty profile and its resources.
- Quick installation of the Liberty profile. You can extract the Liberty profile resource and simply run the install Liberty profile resource job. There is no requirement to run a formal installation tool, such as Installation Manager, to use Liberty profile resources. Liberty profile resources are packaged as compressed files that are ready for use after extraction.
- Support for flexible sharing of resources such as a software development kit (SDK), runtime binary files, server configuration, and application binary files among many server instances.
- Non-destructive update enables easy installation of new versions of any resources. You can switch easily between old and new resources, or run concurrent versions of resources.
- No requirement for agent on target hosts, which reduces administration costs.

Fast, flexible, and simplified application development

WebSphere Application Server includes the following key new features and enhancements to ease, simply and extend the application developer experience:

- The Liberty profile
- Support for OSGi programming model enhancements, including EJB support
- Support for IBM WebSphere SDK Java Technology Edition Version 7.0
- Integrated support for WebSphere Application Server Web 2.0 and Mobile Toolkit
- New migration suite of tools to prepare, analyze, and migrate WebSphere applications and other competitive applications

The Liberty profile

The Liberty profile provides a simplified and lightweight development and application-serving environment for web and OSGi applications. Familiar WebSphere Application Server enterprise qualities of service, such as security and transaction integrity, are enabled as required. Key benefits include, but are not limited to, the following capabilities:

- Free and frictionless download for development purposes
- Ultra lightweight modular runtime environment, with an install size of under 50 MB
- Incredibly fast startup time: under 5 seconds for simple web applications
- Simplified configuration for quick time to productivity
- Availability of WebSphere Application Server Developer Tools as Eclipse plug-ins for broad tooling support
- Platform support for distributed platforms, z/OS, and Mac OS

Because it is composable, each Liberty profile server is usually configured to include only the capabilities that are needed by the applications that it serves. This approach keeps each server small and agile. The Liberty profile includes support for the following capabilities:

- Blueprint
- Java Database Connectivity (JDBC)
- Java Management Extensions (JMX)
- Java Persistence API (JPA)
- JavaServer Faces (JSF)
- JavaServer Pages (JSP)
- Secure Sockets Layer (SSL)
- Servlet
- Web application bundle (WAB)
- Web security

Support for OSGi programming model enhancements, including EJB support

Take advantage of enhanced modular application development by assembling OSGi applications from reusable bundles that contain Enterprise JavaBeans (EJB) assets. The enterprise beans in your OSGi bundles can be developed from scratch, or you can include existing EJB assets and migrate them to use OSGi modularity with minimal code changes. Stateful, stateless, and singleton enterprise beans are supported. Your OSGi application can also contain message-driven beans (MDBs).

You can deploy and configure an OSGi application that contains enterprise beans in a similar way to deploying and configuring a Java Platform, Enterprise Edition (Java EE) enterprise application, by using wsadmin or the administrative console.

Support for IBM WebSphere SDK Java Technology Edition Version 7.0

This IBM Software Development Kit (SDK) provides a full-function SDK for Java that is compliant with the Java Platform, Standard Edition (Java SE) 7 application programming interfaces (APIs). With IBM WebSphere SDK Java Technology Edition Version 7.0, you can develop and deploy Java applications at the Java 7 API level and continue the “write once, run anywhere”Java paradigm at the Java API level. The SDK contains the Java application Runtime Environment and other tools that enable developers to create Java applications.

WebSphere Application Server supports IBM WebSphere SDK Java Technology Edition Version 7.0 as a pluggable JDK. Java 6 is installed with the product and used by default. You can optionally install Java 7 support and then enable it using the managesdk tool.

Integrated support for WebSphere Application Server Web 2.0 and Mobile Toolkit

The WebSphere Application Server Web 2.0 and Mobile Toolkit simplifies the addition of Asynchronous JavaScript and XML (Ajax) rich desktop and mobile user interfaces and Representational State Transfer (REST) Web services to Java Web applications. Web 2.0 capabilities, such as Ajax and REST, help application developers to create more connected, interactive applications, that result in higher customer satisfaction, user productivity, and enhanced decision making. New mobile Ajax components enable developers to create mobile web applications that run on devices such as smartphones and tablets.

New migration suite of tools to prepare, analyze, and migrate WebSphere applications and other competitive applications

Enjoy faster application migrations using the extensive set of tools to help you plan for and implement application migrations. The IBM WebSphere Application Server Migration toolkit is a suite of tools and knowledge collections, provided at no charge, to quickly and cost-effectively migrate to WebSphere Application Server V7, V8.0 or V8.5, whether from a previous version of WebSphere Application Server or competitive application servers including Oracle WebLogic Server, Oracle Application Server, JBoss Application server, and Apache Tomcat. This toolkit provides a single solution for identifying, analyzing, and resolving application code quality and compliance requirements. You can obtain this migration toolkit from the developerWorks IBM WebSphere Application Server Migration Toolkit page.

Integrated tools

WebSphere Application Server Version 8.5 includes:

Development workbench options

Accelerate developer productivity through integrated and optimized developer tooling. Rational Application Developer Version 8.5 provides a complete environment for enterprise development for Java, Java EE, web, web services, SOA, OSGi, and Portal designers and developers. Use this environment to develop, assemble, and deploy your applications to WebSphere Application Server Version 8.5, and test your applications using a rapid-deployment feature in the Version 8.5 runtime environment.

The IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5 is a lightweight set of tools for developing, assembling, and deploying Java EE, OSGi, Web 2.0 and Mobile applications to WebSphere Application Server, including the Liberty profile.

Enhanced resiliency

WebSphere Application Server Version 8.5 provides enhanced user availability and application server resiliency with resiliency enhancements for the service integration bus.

- Improvements to the recovery of messaging engine errors
- Enable the messaging engine to restart after a failure
- Retain the count of failed deliveries after the messaging engine is restarted
- Enhanced memory leak detection and protection

- Improvement to the messaging engine to prevent holding long running database locks
- Improvements to service integration bus performance
- Recovery of the messaging engine configuration from the message store

Improvements to the recovery of messaging engine errors

When a recoverable database error is detected by the high availability (HA) manager, the messaging engine is stopped, and the standby messaging engine is started automatically, if available. The other applications running in the application server are not affected by the failure of the messaging engine.

Enable the messaging engine to restart after a failure

When a messaging engine fails due to recoverable database problems, it is disabled and failed over to another messaging engine in the cluster, if configured for high availability. In the event that the active messaging engine does not responding to the database, the standby messaging engine is able to take ownership of the database because the active messaging engine holds only short duration locks. This action also ensures that only one active messaging engine can access the database at a given time. The disabled messaging engine is automatically enabled after a specific time and is available for failover.

Retain the count of the failed deliveries after the messaging engine is restarted

When a message delivery fails, the messaging engine attempts to redeliver the message repeatedly and the delivery count increases incrementally each time. The redelivery count is persisted to the message store and is made available for the messages even after the messaging engine is restarted.

Enhanced memory leak detection and protection

WebSphere Application Server provides top down pattern-based memory leak detection, prevention, and action by watching for suspect patterns in application code at run time. WebSphere Application Server has some means of protection against memory leaks when stopping or redeploying applications. This product monitors application and module activity and performs diagnostic actions when an application or an individual module stops. This feature helps in increasing application up time with frequent application redeployments without cycling the server.

Improvement to the messaging engine to prevent holding long running database locks

When the messaging engine uses a database as the message store, you can configure the messaging engine to acquire short duration locks on the database; thereby, preventing it from holding long running locks on the database. The new locking mechanism also helps the standby messaging engine to acquire the database ownership when a JVM, hosting the messaging engine in a highly available clustered environment, becomes unresponsive until a specified period of time.

Improvements to service integration bus performance

Improved messaging engine startup time by loading the destinations concurrently in a multi-core architecture. The concurrent loading is possible if the message store is configured with the database which supports parallel reads by multiple threads. The performance improvement is directly proportional to the parallel processing capability of the database and the capacity of the system on which the messaging engine is running.

Recovery of the messaging engine configuration from the message store

You can use the `recoverMEConfig` command to restore the configuration information of the messaging engine from the message store in case the configuration files are lost or corrupted.

Improved operations, security, control, and integration

WebSphere Application Server Version 8.5 includes the following highlights for greater security and for enhanced operations and control for application development, management, and problem determination.

- Improved operations and control
  - Option to select and switch between IBM WebSphere SDK Java Technology Edition Version 7.0 or 6.0
  - Enhancements for Java batch support
  - OSGi Blueprint security improvements
  - Support for OSGi application console
  - Support for the Service Component Architecture (SCA) OASIS programming model implementation
  - Support for Derby 10.8
- Simplified problem determination
  - Cross component trace (XCT) enables correlation of log and trace entries with minimal cost
  - Enhancements to log and trace filtering using application name, request ID, or custom extensions
  - Improved troubleshooting using IBM Support Assistant Data Collector
- Enhanced security for administrative configuration audit tracking
  - Extended repository options

Improved operations and control

Option to select and switch between IBM WebSphere SDK Java Technology Edition Version 7.0 or 6.0

WebSphere Application Server provides support for the IBM WebSphere SDK Java Technology Edition Version 7.0 as an optional pluggable Java development kit (JDK). Java 6 is installed with the product and used by default. Use the managesdk tool to optionally install and enable Java 7. You can optionally switch between using Java 6 or Java 7 to best meet your business needs.

Enhancements for Java batch support

WebSphere Application Server provides powerful new batch application features. Enhancements include support for parallel batch processing to reduce batch job elapsed time, memory-overload protection for batch workloads, integration of workload scheduler to submit batch work from scheduler products like Tivoli Workload Scheduler, and numerous xJCL and programming model improvements to increase application development flexibility.

OSGi Blueprint security improvements

You can configure bean security in the Blueprint XML file of your OSGi applications, so that the methods of the bean can be accessed only by users that are assigned a specified role. You can configure bean-level security, so that a single role is associated with all the methods of the bean, or you can configure method-level security, where different roles are associated with specific methods.

Support for the OSGi application console

The WebSphere Application Server administrative console provides pages that you can use to examine or debug a specific set of OSGi bundles running on an application server. You can, for example, view and explore the package and service dependencies between the bundles in an OSGi application.

Support for the Service Component Architecture (SCA) OASIS programming model implementation

WebSphere Application Server Version 8.5 adds support for the SCA OASIS programming model implementation. The product provides partial support for the following OASIS specifications:

- OASIS SCA Assembly Model Specification 1.1

- OASIS SCA Policy Framework Specification 1.1

  The product supports OASIS policy attachment, but does not support OASIS policy set definitions.
- OASIS SCA-Bindings specifications:
  - SCA JMS Binding Specification 1.1
  - SCA Web Service Binding Specification 1.1
- OASIS SCA-J Common Annotations and APIs Specification 1.1

This product also supports EJB binding, POJO, JAXB and SDO as data types.

Support for Derby 10.8

Connect your applications to the latest versions of a wide array of industry-leading databases to enable maximum deployment flexibility. WebSphere Application Server Version 8.5 has been tested with and includes the Derby 10.8 database.

Simplified problem determination

Cross component trace (XCT) enables correlation of log and trace entries with minimal cost

Use the cross component trace facility to identify the root cause of problems across components. XCT annotates log and trace entries so that log entries that are related to a request that is serviced by more than one thread, process, or even server are identified as belonging to the same unit of work. This enhancement enables administrators and support teams to follow the flow of a request from end-to-end as it traverses thread or process boundaries.

Enhancements to log and trace filtering using application name, request ID, or custom extensions

Use the High Performance Extensible Logging (HPEL) LogViewer command-line tool to filter records based on the content of log and trace record extensions. The application server automatically creates an appName extension for each log and trace record related to a Java Platform, Enterprise Edition (Java EE) application, indicating the name of that application. The application server also automatically creates a requestId extension when cross component trace is enabled for each log and trace record created during the processing of certain types of requests (for example HTTP or JMS requests), indicating the unique ID of that request. You can add custom extensions to HPEL log and trace entries using the LogRecordContext API.

Improved troubleshooting using IBM Support Assistant Data Collector

The IBM Support Assistant Data Collector for WebSphere Application Server tool focuses on automatic collection of problem data and is now included as part of this product. It also provides symptom analysis support for the various categories of problems encountered by IBM software products. Information pertinent to a type of problem is collected to help identify the origin of the problem under investigation. The tool assists customers by reducing the amount of time it takes to reproduce a problem with the proper RAS tracing levels set, as well as by reducing the effort required to send the appropriate log information in to IBM Support.

Extended repository options

Enhanced security for administrative configuration audit tracking

WebSphere Application Server Version 8.5 enables you to track changes made to your application server configuration using checkpoints made through the extended repository service. A full checkpoint is a complete copy of the entire configuration repository. A delta checkpoint is a subset snapshot of the configuration repository that is made when you change a product configuration. Use a checkpoint to restore the configuration repository back to a prior state. To determine what has changed in the configuration, you can extract from a delta checkpoint to obtain the before and after versions of the files that were saved.

Chapter 29. Overview and new features for administering applications and their environments

Use the links provided in this topic to learn about the administrative features.

What is new for administrators

This topic provides an overview of new and changed features of system administration.

"Introduction: System administration"

This topic describes the administration of the product and the applications that run on it.

See also "Introduction: Environment" on page 994 and Introduction: Variables.

What is new for administrators

This topic highlights what is new or changed, for users who are going to customize, administer, monitor, and tune production server environments. It also addresses those who are going to deploy and operate applications.

- Migrating product configurations with migration tools
  - You can use the z/OS Migration Management Tool to collect the information and generate the Job Control Language (JCL) jobs (CNTL and DATA datasets) for migrating a stand-alone application server, deployment manager, or federated node. You must create these jobs before you can begin the actual migration.
  - You can use the `zmmt` command with a response file to generate the JCL jobs (CNTL and DATA datasets) for migrating a stand-alone application server, deployment manager, or federated node. You must create these jobs before you can begin the actual migration.
- Scenario 2: Migrating a job manager profile and its registered set of servers using the z/OS Configuration Migration Management Tool

  You can migrate job manager profiles and the registered set of servers from Version 7.0 or above to Version 8.5.

Introduction: System administration

You can administer your WebSphere Application Server product through scripts, command line tools, the administrative console, or the Java programming interface. You administer server processes, topological units referenced as nodes and cells, and the configuration repository where configuration information is stored in Extensible Markup Language (XML) files.

Note: If you would prefer to browse PDF versions of this documentation using your Adobe Reader, see the **System Administration** PDF files available from www.ibm.com/software/webservers/appserv/infocenter.html.

A variety of tools, processes, and configuration files are provided for administering the product:

- **Console**

  The administrative console is a graphical interface that provides many features to guide you through deployment and systems administration tasks. Use it to explore available management options.

  For more information, refer to "Introduction: Administrative console" on page 990.
- **Scripting**

  The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. You can also submit scripting language programs to run in batch mode. The wsadmin tool is intended for production environments and unattended operations.

For more information, refer to “Introduction: Administrative scripting (wsadmin).”

- **Command line tools**

  Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

  For more information, refer to “Introduction: Administrative commands” on page 991.
- **Programming**

  The product supports a Java programming interface for developing administrative programs. All of the administrative tools supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification.

  For more information, refer to “Introduction: Administrative programs” on page 991.
- **Data**

  Product configuration data resides in XML files that are manipulated by the previously-mentioned administrative tools.

  For more information, refer to “Introduction: Administrative configuration data” on page 992.
- IBM i

  Product library, directories, subsystem, job queue, job description, and output queues

  An IBM i platform uses different configurations than installations of WebSphere Application Server on other platforms.

  For more information, refer to “Product library, directories, subsystem, job queue, job description, and output queues” on page 992.

Introduction: Administrative console

The administrative console is a graphical interface that allows you to manage your applications and perform system administration tasks for your WebSphere Application Server environment. The administrative console runs in your web browser.

Your actions in the console modify a set of XML configuration files.

You can use the administrative console to perform tasks such as:

- Add, delete, start, and stop application servers
- Deploy new applications to a server
- Start and stop existing applications, and modify certain configurations
- Add and delete Java Platform, Enterprise Edition (Java EE) resource providers for applications that require data access, mail, URLs, and so on
- Configure product security, including access to the administrative console
- Collect data for performance and troubleshooting purposes
- Find the product version information. It is located on the front page of the console.

Starting and logging off the administrative console helps you begin using the console so that you can explore the available options. See also the **Reference > Settings** section of the information center navigation. It lists the settings or properties you can configure.

Introduction: Administrative scripting (wsadmin)

The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language.

About this task

The wsadmin tool is intended for production environments and unattended operations. You can use the wsadmin tool to perform the same tasks that you can perform using the administrative console.

The following list highlights the topics and tasks available with scripting:

Procedure

- Getting started with scripting Provides an introduction to WebSphere Application Server scripting and information about using the wsadmin tool. Topics include information about the scripting languages and the scripting objects, and instructions for starting the wsadmin tool.
- Using the Jython script library The script library provides Jython script procedures to assist in automating your environment. Use the sample scripts to manage applications, resources, servers, nodes, and clusters. You can also use the script procedures as examples to learn the Jython syntax.
- Deploying applications Provides instructions for deploying and uninstalling applications. For example, stand-alone Java archive files and web application archive (WAR) files, the administrative console, remote enterprise archive (EAR) files, file transfer applications, and so on.
- Managing deployed applications Includes tasks that you perform after the application is deployed. For example, starting and stopping applications, checking status, modifying listener address ports, querying application state, configuring a shared library, and so on.
- Configuring servers Provides instructions for configuring servers, such as creating a server, modifying and restarting the server, configuring the Java virtual machine, disabling a component, disabling a service, and so on.
- Configuring connections to web servers Includes topics such as regenerating the plug-in, creating new virtual host templates, modifying virtual hosts, and so on.
- Managing servers Includes tasks that you use to manage servers. For example, stopping nodes, starting and stopping servers, querying a server state, starting a listener port, and so on.
- Configuring security Includes security tasks such as enabling and disabling security.
- Configuring data access Includes topics such as configuring a Java DataBase Connectivity (JDBC) provider, defining a data source, configuring connection pools, and so on.
- Configuring messaging Includes topics about messaging, such as Java Message Service (JMS) connection, JMS provider, WebSphere queue connection factory, MQ topics, and so on.
- Configuring mail, URLs, and resource environment entries Includes topics such as mail providers, mail sessions, protocols, resource environment providers, referenceables, URL providers, URLs, and so on.
- Dynamic caching Includes caching topics, for example, creating, viewing and modifying a cache instance.
- Troubleshooting Provides information about how to troubleshoot using scripting. For example, tracing, thread dumps, profiles, and so on.
- Obtaining product information Includes tasks such as querying the product identification.
- Scripting reference material Includes all of the reference material related to scripting. Topics include the syntax for the wsadmin tool and for the administrative command framework, explanations and examples for all of the scripting object commands, the scripting properties, and so on.

Introduction: Administrative commands

Command line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

See **Reference** > **Command-line utilities** in the information center navigation area for the names and syntax of all the commands that are available with the product. A subset of these commands is particular to system administration purposes.

Introduction: Administrative programs

The Java Management Extensions (JMX) specification allows you to write Java programs to administer WebSphere Application Server.

The product supports a Java programming interface for developing administrative programs. All of the administrative tools supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the WebSphere Application Server administrative tools. You can also extend the basic WebSphere Application Server administrative system to include your own managed resources.

Introduction: Administrative configuration data

WebSphere Application Server configuration data is kept in files. All administrative actions that you perform involve changes to these files.

Administrative tasks typically involve defining new configurations of the product or performing operations on managed resources within the environment. WebSphere Application Server configuration data is kept in files. Because all product configuration involves changing the content of those files, it is useful to know the structure and content of the configuration files.

The WebSphere Application Server product includes an implementation of the Java Management Extension (JMX) specification. All operations on managed resources in the product go through JMX functions. This setup means a more standard framework underlying your administrative operations as well as the ability to tap into the systems management infrastructure programmatically.

Product library, directories, subsystem, job queue, job description, and output queues

An IBM i platform uses different configurations than installations of WebSphere Application Server on other platforms.

This topic describes the product library, directories, subsystem, job queue, job description, and output queues that WebSphere Application Server uses on the IBM i platform.

Product library and directories

In a default installation, all editions of WebSphere Application Server for IBM i use the following library and directories:

QWAS85 library
Product library

`/QIBM/UserData/WebSphere/AppServer/V85` directory
Root directory for user data that is common to all editions of the product.

`/QIBM/ProdData/WebSphere/AppServer/V85/Base` directory
Root directory containing product data that is shared by all WebSphere Application Server profiles. Additionally, the master copies of the files, which make up a profile, are stored here. Do not modify files under this directory structure.

`/QIBM/UserData/WebSphere/AppServer/V85/Base` directory
Root directory for profile information. If the -profilePath option is not specified when creating a profile, the profile is created under this path.

Subsystem

All editions of the product for IBM i use the QWAS85 subsystem. The subsystem description for QWAS85 is located in the QWAS85 product library. The application server, node agent and deployment manager processes run in this subsystem.

Job queue

All editions of the product for IBM i use the QWASJOBQ job queue. The job queue is in the QWAS85 product library. The application server, node agent, and deployment manager processes are all submitted to this job queue when you invoke the startServer, startNode, or startManager scripts. The QWASJOBQ job queue is associated with the QWAS85 subsystem.

Job description

All editions of the product for IBM i use the QWASJOBD job description. The job description is in the QWAS85 product library. The application server, node agent, and deployment manager processes use this job description when submitted to run in the QWAS85 subsystem.

The QWASJOBD job description has the following default settings:

```
                  Display Job Description
                                                     System:   MYISERIES
 Job description:   QWASJOBD       Library:   QWAS85

 User profile . . . . . . . . . . . . . . . . . . :   QEJBSVR
 CL syntax check  . . . . . . . . . . . . . . . . :   *NOCHK
 Hold on job queue  . . . . . . . . . . . . . . . :   *NO
 End severity . . . . . . . . . . . . . . . . . . :   30
 Job date . . . . . . . . . . . . . . . . . . . . :   *SYSVAL
 Job switches . . . . . . . . . . . . . . . . . . :   00000000
 Inquiry message reply  . . . . . . . . . . . . . :   *RQD
 Job priority (on job queue)  . . . . . . . . . . :   5
 Job queue  . . . . . . . . . . . . . . . . . . . :   QWASJOBQ
   Library  . . . . . . . . . . . . . . . . . . . :   QWAS85
 Output priority (on output queue)  . . . . . . . :   5
 Printer device . . . . . . . . . . . . . . . . . :   *USRPRF
 Output queue . . . . . . . . . . . . . . . . . . :   QWASOUTQ
   Library  . . . . . . . . . . . . . . . . . . . :   QWAS85
 Message logging:
   Level  . . . . . . . . . . . . . . . . . . . . :   4
   Severity . . . . . . . . . . . . . . . . . . . :   0
   Text . . . . . . . . . . . . . . . . . . . . . :   *SECLVL
 Log CL program commands  . . . . . . . . . . . . :   *NO
 Accounting code  . . . . . . . . . . . . . . . . :   *USRPRF
 Print text . . . . . . . . . . . . . . . . . . . :   *SYSVAL

 Routing data . . . . . . . . . . . . . . . . . . :   QCMDI

 Request data . . . . . . . . . . . . . . . . . . :   *NONE

 Device recovery action . . . . . . . . . . . . . :   *SYSVAL
 Time slice end pool  . . . . . . . . . . . . . . :   *SYSVAL
 Job message queue maximum size . . . . . . . . . :   *SYSVAL
 Job message queue full action  . . . . . . . . . :   *WRAP
 Allow multiple threads . . . . . . . . . . . . . :   *YES
 Initial ASP group  . . . . . . . . . . . . . . . :   *NONE
 Spooled file action  . . . . . . . . . . . . . . :   *SYSVAL
 Text . . . . . . . . . . . . . . . . . . . . . . :   WEBSPHERE JOB DESCRIPTION
```

Use the **Request data** parameter to specify information to start a WebSphere Application Server profile when the QWAS85 subsystem is started. For more information on how to start an application server when the subsystem starts see Starting the application server environment in the QWAS85 subsystem. You can modify the job description if the values do not meet your requirements. However, you must not change the **Allow multiple threads** value. Because the product processes contain a Java virtual machine that is multithreaded, this value must be *YES.

Output queues

All editions of the product for IBM i use the QWASOUTQ output queue. The output queue is in the QWAS85 product library. By default, this output queue stores job logs for these processes:
- The application server process
- The node agent process
- The deployment manager process

This queue also stores other spooled files. The public authority to the output queue is `*USE`. This setting ensures that the user profile that is specified for the USER parameter (on the SBMJOB command) has the necessary authority to the output queue when it submits the WebSphere Application Server job to the job queue. The default user profile is QEJBSVR.

You might want to clear the output queue periodically if the number or size of the spooled files becomes large.

Introduction: Environment

Your WebSphere Application Server product environment includes web server plug-ins, WebSphere Application Server variables, and other data objects. Configure values for settings in these categories using the Environment section of the administrative console.

Web servers

In the WebSphere Application Server product, an application server works with a web server to handle requests for web applications. The application server and web server communicate using a WebSphere HTTP plug-in for the web server.

Variables

Variables come in many varieties. They are used to control settings and properties relating to the server environment. The three main types of variables that are important for you to understand are environment variables, WebSphere variables, and custom properties.

For more information, refer to Introduction: Variables.

Introduction: Application servers

Application servers provide the core functionality of the WebSphere Application Server product family. Application servers extend the ability of a web server to handle Web application requests, and much more. An application server enables a server to generate a dynamic, customized response to a client request.

Introduction: Application servers

An application server is a Java Virtual Machine (JVM) that runs user applications. The application server collaborates with the web server to return a dynamic, customized response to a client request. The client request can consist of servlets, JavaServer Pages (JSP) files, and enterprise beans, and their supporting classes.

For example, a user at a web browser visits a company website:
1. The user requests access to data in a database.
2. The user request flows to the web server.
3. The web server determines that the request involves an application containing resources not handled directly by the web server (such as servlets). It forwards the request to one of its application servers on which the application is running.
4. The invoked application then processes the user request. For example:

- An application servlet prepares the user request for processing by an enterprise bean that performs the database access.
- The application produces a dynamic web page containing the results of the user query.

5. The application server collaborates with the web server to return the results to the user at the web browser.

When you install the product, a default application server, named server1, is automatically created. You can use the administrative console to manage this server.

You can use the administrative console or command-line tools to create additional application servers that can either be separately configured processes or nearly identical clones. You must either use command-line tools to manage these additional servers, set up an administrative console for each server, or configure an administrative agent to provide a single interface to all of your servers, including the original base server. An administrative agent makes it easier to more fully administer these unfederated application servers.

gotcha: If you create additional application servers, only use one server to modify and save configurations. There is no coordination of configuration setting between the different servers and if you modify and save configurations on multiple servers, your data might become corrupted.

You can improve system performance if you configure some of your application servers, such that each of their components are dynamically started as they are needed, instead of letting all of these components automatically start when the server starts. Selecting this option can improve server startup time, and reduce the memory footprint. Starting components as they are needed is most effective if all of the applications that are deployed on the application server are of the same type. For example, using this option works better if all of your applications are web applications that use servlets, and JavaServer Pages (JSP). This option works less effectively if your applications use servlets, JSPs and Enterprise JavaBeans (EJB).

You can also perform the following tasks to enhance the operation of an application server:

- Configure transport chains to provide network protocol support for I/O operations within an application server environment.
- Add an interface to an application server to define a hook point that runs when the server starts and shuts down.
- Define command-line information that passes to a server when it starts or initializes.
- Tune the application server.
- Enhance the performance of the application server JVM.
- Configure an Object Request Broker (ORB) for RMI/IIOP communication.

Asynchronous messaging

The product supports asynchronous messaging based on the Java Message Service (JMS) of a JMS provider that conforms to the JMS specification Version 1.1.

The JMS functions of the default message service that is provided with the product are served by one or more messaging engines (in a service integration bus) that runs within application servers.

Generic Servers

A generic server is a server that is managed in the WebSphere administrative domain, although it is not a server that is supplied by the product. The generic server can be any server or process that is necessary to support the product environment.

Introduction: Web servers

An application server works with a web server to handle requests for dynamic content, such as servlets, from web applications. A web server uses a web server plug-in to establish and maintain persistent HTTP and HTTPS connections with an application server.

The Supported Hardware and Software web page provides the most current information about supported web servers.

Implementing a web server plug-in describes how to set up your web server and Web server plug-in environment and how to create a web server definition. The web server definition associates a web server with an application server. After you create a web server definition, you can use the administrative console to perform the following functions for that web server:

- Check the status of the web server
- Generate a plug-in configuration file for that web server.

You can not propagate a plug-in configuration file for a non-IBM HTTP Server. You must manually install an updated plug-in configuration file on that web server.

After you set up your web server and web server plug-in, whenever you deploy a web application, you must specify a web server as the deployment target that serves as a router for requests to the Web application. The configuration settings in the plug-in configuration file (plugin-cfg.xml) for each web server are based on the applications that are routed through that web server. If the web server plug-in configuration service is enabled, a web server plug-in's configuration file is automatically regenerated whenever a new application is associated with that web server.

Note: Before starting the web server, make sure you are authorized to run any Application Response Measurement (ARM) agent associated with that web server.

Refer to your web server documentation for information on how to administer that web server. For tips on tuning your web server plug-in, see Web server plug-in tuning tips.

Mail, URLs, and other J2EE resources

This topic describes the supported resources that are defined by Java Platform, Enterprise Edition (Java EE).

The product supports all of the resources defined by the Java EE. It adds the following resources in support of service extensions:

- Schedulers
- Work managers
- Object pools

Data access (JDBC and J2C)

The J2EE Connector architecture defines a standard architecture that enables the integration of various enterprise information systems (EIS) with application servers and enterprise applications. It defines a standard resource adapter used by a Java application to connect to an EIS. This resource adapter can plug into the application server and, through the Common Client Interface (CCI), provide connectivity between the EIS, the application server, and the enterprise application.

For more information, refer to “Data access resources” on page 997.

Messaging

The product supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The base JMS support enables the product applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

For more information, refer to “Messaging resources” on page 998.

Mail

Using JavaMail API, a code segment can be embedded in any Java EE application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

For more information, refer to JavaMail API.

URLs

Java EE applications can use URLs as resources in the same way other Java EE resources, such as JDBC and JavaMail, are used.

For more information, refer to “URLs” on page 342.

Resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

For more information, see Configuring new resource environment entries to map logical environment resource names to physical names.

Data access resources

These topics provide information about accessing data resources.

The connection between an enterprise application and an enterprise information system (EIS) is accomplished through the use of EIS-provided resource adapters, which are plugged into the application server. The resource adapter plays a central role in the integration and connectivity between an EIS and an application server. It serves as the point of contact between application components, application servers, and enterprise information systems. A resource adapter must communicate with other components based on well-defined contracts that are specified by the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA).

Generic inflow context enables a resource adapter to control the execution context of the Work instances it submits to the application server. By submitting a Work instance that implements the WorkContextProvider interface, a resource adapter can provide various types of context to the WebSphere Application Server. If the application server supports the provided context types, the generic work context mechanism sets the work contexts as the execution context of the Work instance. The context remains effective during the execution the Work instance.

Security inflow context uses generic work context by enabling a resource adapter to establish security information in the execution context of the Work instances that it submits to the application server. By submitting a Work instance that provides context types by implementing the new standardized SecurityContext interface, the application can establish and set an execution context containing the security identities and credentials for a Work instance. The identities and credentials remain effective during the execution of the Work instance, ensuring secure message delivery to Java EE message endpoints.

WebSphere Application Server supports work context types that implement the new standardized SecurityContext, TransactionContext and HintsContext interfaces. The generic inflow context mechanism accepts implementations of the HintsContext interface, but the application server does not act upon these implementations of the HintsContext interface. The security inflow context mechanism does not map user identities from the EIS domain to identities in an application server domain. Identities provided by implementations of SecurityContext must be in a security domain of application server.

Consult the following concept, reference, and task files for more overview information.

Messaging resources

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages. Applications can use point-to-point and publish/subscribe messaging. These styles of messaging can be used in the following ways: one-way; request and response; one-way and forward.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages. Message-driven beans can be configured as listeners on a Java EE Connector Architecture (JCA) 1.5 or 1.6 resource adapter or against a listener port.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider)
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider)
- Third-party messaging providers that implement either a JCA Version 1.5 or 1.6 resource adapter or the ASF component of the JMS Version 1.0.2 specification

Your applications can use messaging resources from any of these JMS providers. The choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you might already have a messaging infrastructure based on WebSphere MQ. In this case, you can either connect directly by using the WebSphere MQ messaging provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is a logical choice. If your business also uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominately WebSphere MQ network, choose the WebSphere MQ messaging provider. To administer a third-party messaging provider, you use either the resource adaptor (for a Java EE Connector Architecture (JCA) 1.5-compliant or 1.6-compliant messaging provider) or the client (for a non-JCA messaging provider) that is supplied by the third party.

For more information, see “Introduction: Messaging resources” on page 1046.

Chapter 30. Overview and new features for securing applications and their environment

Use the links provided in this topic to learn more about the security infrastructure.

What is new for security specialists

This topic provides an overview of new and changed features in security.

“Security”

This topic describes how IBM WebSphere Application Server provides security infrastructure and mechanisms to protect sensitive Java Platform, Enterprise Edition (Java EE) resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

“Security planning overview” on page 1009

Several communication links are provided from a browser on the Internet, through web servers and product servers, to the enterprise data at the back-end. This topic examines some typical configurations and common security practices. WebSphere Application Server security is built on a layered security architecture. This section also examines the security protection offered by each security layer and common security practice for good quality of protection in end-to-end security.

Samples

The Samples documentation offers:

- **Login - Form Login**

  The Form Login Sample demonstrates a very simple example of how to use the login facilities for WebSphere Application Server to implement and configure login applications. The Sample uses the Java Platform, Enterprise Edition (Java EE) form-based login technology to customize the look and feel of the login screens. It uses servlet filters to log the user information and the date information. The Sample finishes the session by using the form-based logout function, an IBM extension to the Java EE specification.
- **Login - JAAS Login**

  The JAAS Login Sample demonstrates how to use the Java Authentication and Authorization Service (JAAS) with WebSphere Application Server. The Sample uses server-side login with JAAS to authenticate a real user to the WebSphere security run time. Based upon a successful login, the WebSphere security run time uses the authenticated Subject to perform authorization checks on a protected stateless session enterprise bean. If the Sample runs successfully, it displays all the principals and public credentials of the authenticated user.

Security

The following information provides an overview of security in WebSphere Application Server.

IBM WebSphere Application Server provides security infrastructure and mechanisms to protect sensitive Java 2 Platform, Enterprise Edition (J2EE) resources and administrative resources. It also addresses enterprise end-to-end security requirements on:

- Authentication
- Resource access control
- Data integrity
- Confidentiality
- Privacy

- Secure interoperability

IBM WebSphere Application Server security is based on industry standards and has an open architecture that ensures secure connectivity and interoperability with Enterprise Information Systems (EIS) including:

- Database 2 (DB2)
- CICS
- MQ Series
- Lotus Domino
- IBM Directory

WebSphere Application Server also supports other security providers including:

- IBM Tivoli Access Manager (Policy Director)
- Reverse secure proxy server including WebSEAL

Based on industry standards

IBM WebSphere Application Server provides a unified, policy-based, and permission-based model for securing web resources, web service endpoints, and enterprise JavaBeans according to J2EE specifications. Specifically, WebSphere Application Server complies with Java EE 6 specification and has passed the J2EE Compatibility Test Suite.

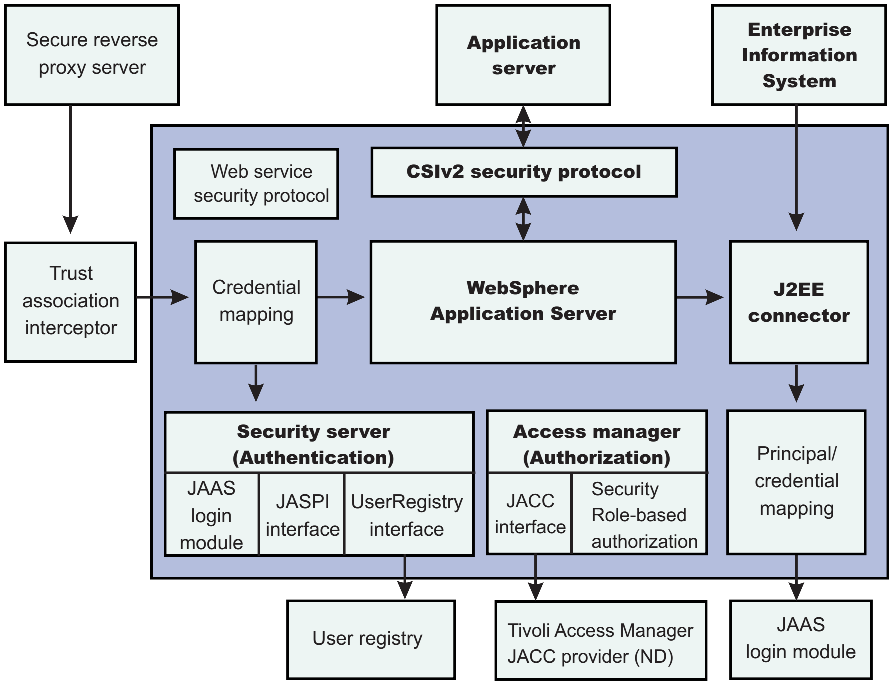
WebSphere Application Server security is a layered architecture built on top of an operating system platform, a Java virtual machine (JVM), and Java 2 security. This security model employs a rich set of security technology including the:

- Java 2 security model, which provides policy-based, fine-grained, and permission-based access control to system resources.
- Common Secure Interoperability Version 2 (CSIv2) security protocol, in addition to the Secure Authentication Services (SAS) security protocol. Both protocols are supported by prior WebSphere Application Server releases. CSIv2 is an integral part of the J2EE 1.4 specification and is essential for interoperability among application servers from different vendors and with enterprise CORBA services.
- Java Authentication and Authorization Service (JAAS) programming model for Java applications, servlets, and enterprise beans.
- J2EE Connector architecture for plugging in resource adapters that support access to Enterprise Information Systems.

The standard security models and interfaces that support secure socket communication, message encryption, and data encryption are the Java Secure Socket Extension (JSSE) and the Java Cryptographic Extension (JCE).

Open architecture paradigm

An application server plays an integral part in the multiple-tier enterprise computing framework. IBM WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components. Plug-in points are based on standard J2EE specifications wherever applicable.


Secure reverse proxy server
Application server
Enterprise Information System
Web service security protocol
CSIv2 security protocol
Trust association interceptor
Credential mapping
WebSphere Application Server
J2EE connector
Security server (Authentication)
JAAS login module
JASPI interface
UserRegistry interface
Access manager (Authorization)
JACC interface
Security Role-based authorization
Principal/ credential mapping
User registry
Tivoli Access Manager JACC provider (ND)
JAAS login module

The dark blue shaded background indicates the boundary between the WebSphere Application ServerVersion 8.5 and other business application components.

IBM i WebSphere Application Server provides Simple WebSphere Authentication Mechanism (SWAM), Lightweight Third Party Authentication (LTPA), and Kerberos as the authentication mechanisms. Exactly one user registry implementation can be configured to be the active user registry of WebSphere Application Server security domain. WebSphere Application Server provides the following user registry implementations: UNIX, Windows, and IBM i local OS and Lightweight Directory Access Protocol (LDAP). It also provides file-based and Java database connectivity (JDBC)-based user registry reference implementations. It supports a flexible combination of authentication mechanisms and user registries. SWAM is simple to configure and is useful for a single application server environment. It is possible to use SWAM in a distributed environment if identity assertion is enabled. The identity assertion feature is available only on the CSIv2 security protocol.

Note: SWAM was deprecated in a previous release of WebSphere Application Server, and will be removed in a future release.

IBM i The LTPA authentication mechanism is designed for all platforms security. Downstream servers can validate the security token. It also supports setting up a trust association relationship with reverse secure proxy servers and single sign-on (SSO), which is discussed later. Besides the combination of LTPA and LDAP or Custom user registry interface, Version 6.x or higher supports LTPA with a LocalOS user registry interface. The new configuration is particularly useful for a single node with multiple application servers. It can function in a distributed environment if the local OS user registry implementation is a centralized user registry, such as Windows Domain Controller, or can be maintained in a consistent state on multiple nodes.

IBM i WebSphere Application Server supports the J2EE Connector architecture and offers container-managed authentication. It provides a default Java 2 Connector (J2C) principal and credential mapping module that maps any authenticated user credential to a password credential for the specified Enterprise Information Systems (EIS) security domain. The mapping module is a special JAAS login module designed according to the Java 2 Connector and JAAS specifications. Other mapping login modules can be plugged in.

Web Services Security

WebSphere Application Server enables you to secure web services based upon the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Version 1.1 specification. These standards address how to provide protection for messages exchanged in a web service environment. The specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

Trust associations

Trust association enables you to integrate third-party security servers with IBM WebSphere Application Server security. More specifically, a reverse proxy server can act as a front-end authentication server while the WebSphere Application Server applies its own authorization policy onto the resulting credentials that are passed by the proxy server. The reverse proxy server applies its authentication policies to every web request that is dispatched to WebSphere Application Server. The products that implement trust association interceptors (TAI) include:

- IBM Tivoli Access Manager for e-business
- WebSEAL
- Caching Proxy

For more information on using trust association, refer to Trust associations.

Security attribute propagation

Security attribute propagation enables WebSphere Application Server to transport security attributes from one server to another in your configuration. Security attributes include authenticated subject contents and security context information. WebSphere Application Server can obtain these security attributes from either:

- An enterprise user registry that queries static attributes
- A custom login module that can query static or dynamic attributes

Security attribute propagation provides propagation services using Java serialization for any objects that are contained in the subject. For more information on using security attribute propagation, refer to Security attribute propagation.

Single sign-on interoperability mode

In WebSphere Application Server, the interoperability mode option enables Single Sign-on (SSO) connections between WebSphere Application Server version 6.1.x or later to interoperate with previous versions of the application server. When you select this option, WebSphere Application Server adds the old-style LtpaToken into the response so that it can be sent to other servers that work only with this token type. This option applies only when the web inbound security attribute propagation option is enabled. For more information on single sign-on, refer to Implementing single sign-on to minimize web user authentications

IBM i

Security for J2EE resources using web containers and EJB containers

Each container provides two kinds of security: *declarative security* and *programmatic security*. In declarative security, the security structure of an application, including data integrity and confidentiality, authentication requirements, security roles, and access control, is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere Application Server maintains a J2EE security policy, including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control. When declarative security alone is not sufficient to express the security model of an application, the application code can use programmatic security to make access decisions. The application programming interface (API) for programmatic security consists of two methods of the Enterprise JavaBeans (EJB) EJBContext interface (`isCallerInRole`, `getCallerPrincipal`) and three methods of the servlet HttpServletrequest interface (`isUserInRole`, `getUserPrincipal`, `getRemoteUser`).

Web security

When a security policy is specified for a web resource and IBM WebSphere Application Server security is enforced, the web container performs access control when the resource is requested by a web client. The web container challenges the web client for authentication data if none is present according to the specified authentication method, ensures that the data constraints are met, and determines whether the authenticated user has the required security role. WebSphere Application Server supports the following login methods:

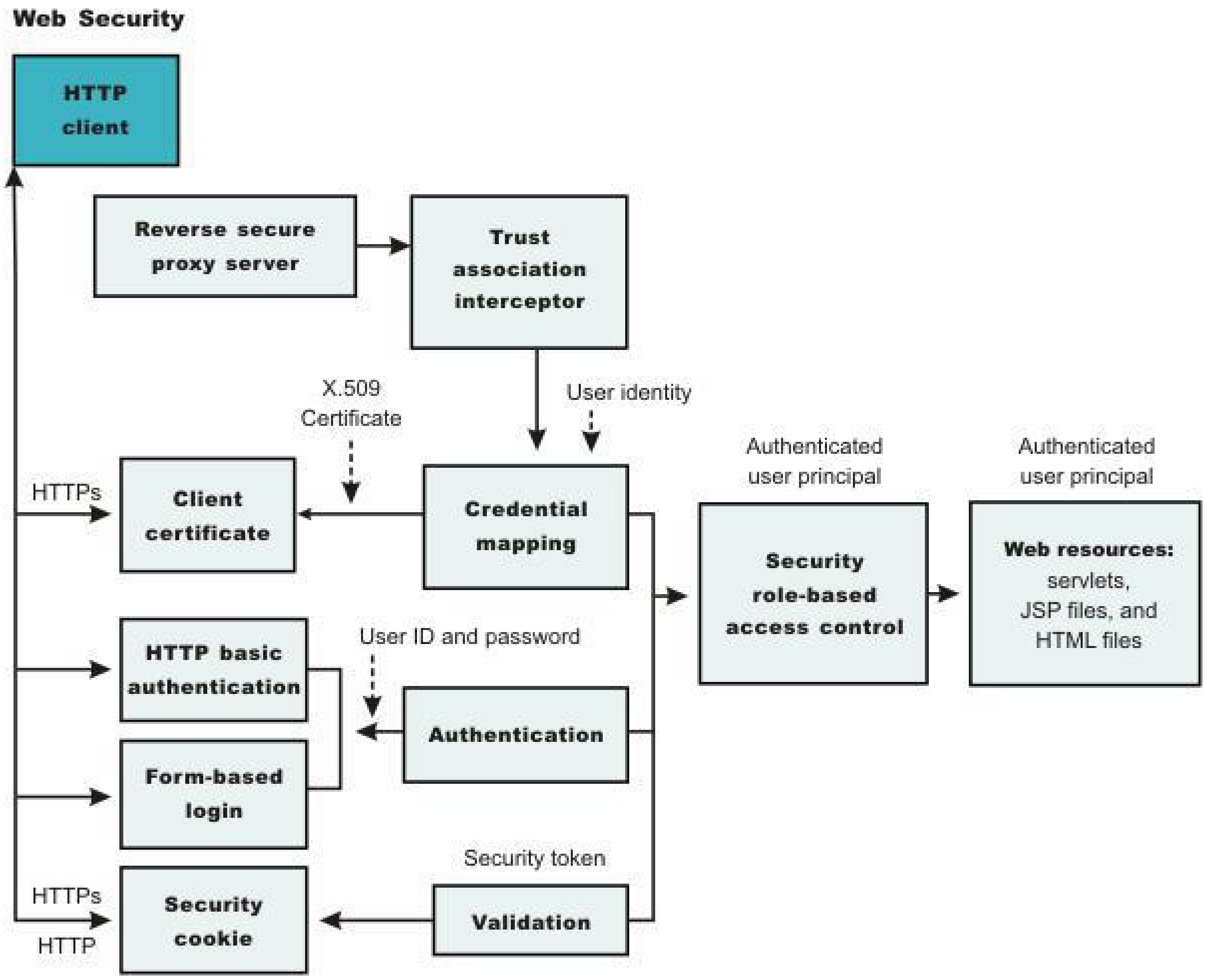
- HTTP basic authentication
- HTTPS client authentication
- Form-based Login
- Simple and Protected GSS-API Negotiation (SPNEGO) token

Mapping a client certificate to a WebSphere Application Server security credential uses the UserRegistry implementation to perform the mapping.

IBM i On WebSphere Application Server, Express®, the local OS user registry does not support the mapping function.

IBM i When the LTPA authentication mechanism is configured and single sign-on (SSO) is enabled, an authenticated client is issued a security cookie, which can represent the user within the specified security domain.

It is recommended that you use Secure Sockets Layer (SSL) to protect the security cookie or Basic Authentication information from being intercepted and replayed. When a trust association is configured, WebSphere Application Server can map an authenticated user identity to security credentials based on the trust relationship established with the secure reverse proxy server.


Web Security
HTTP client
Reverse secure proxy server
Trust association interceptor
X.509 Certificate
User identity
Authenticated user principal
Authenticated user principal
HTTPs
Client certificate
Credential mapping
Security role-based access control
Web resources: servlets, JSP files, and HTML files
HTTP basic authentication
User ID and password
Authentication
Form-based login
Security token
HTTPs
HTTP
Security cookie
Validation

When considering web security collaborators and EJB security collaborators:

1. The web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested Servlet or JSP file if the user principal has one of the required security roles. Servlets and JSP files can use the HttpServletRequest methods: `isUserInRole`, `getUserPrincipal`, and `getRemoteUser`. As an example, the administrative console uses the `isUserInRole` method to determine the proper set of administrative functionality to expose to a user principal.
2. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the EJBContext methods `isCallerInRole` and `getCallerPrincipal`. EJB code also can use the JAAS programming model to perform JAAS login and `WSSubject doAs` and `doAsPrivileged` methods. The code in the `doAs` and `doAsPrivileged PrivilegedAction` block executes under the Subject identity. Otherwise, the EJB method executes under either the `RunAs` identity or the caller identity, depending on the `RunAs` configuration.

EJB security

When security is enabled, the EJB container enforces access control on EJB method invocation. The authentication takes place regardless of whether a method permission is defined for the specific EJB method.

A Java application client can provide the authentication data in several ways. Using the `sas.client.props` file, a Java client can specify whether to use a user ID and password to authenticate or to use an SSL client certificate to authenticate. The client certificate is stored in the key file or in the hardware cryptographic card, as defined in a `sas.client.props` file. The user ID and password can be optionally defined in the `sas.client.props` file.

At run time, the Java client can either perform a programmatic login or perform a *lazy authentication*.

In lazy authentication when the Java client is accessing a protected enterprise bean for the first time, the security run time tries to obtain the required authentication data. Depending on the configuration setting in `sas.client.props` file the security runtime either looks up the authentication data from this file or prompts the user. Alternatively, a Java client can use programmatic login. WebSphere Application Server supports the JAAS programming model and the JAAS login (LoginContext) is the recommended way of programmatic login. The login_helper request_login helper function is deprecated in Version 6.x and Version 8.5. Java clients programmed to the login_helper APT can run in this version.

The EJB security collaborator enforces role-based access control by using an access manager implementation.

An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the EJBContext methods isCallerInRole and getCallerPrincipal. EJB code also can use the JAAS programming model to perform JAAS login and WSSubject doAs and doAsPrivileged methods. The code in the doAs and doAsPrivileged PrivilegedAction block executes under the Subject identity. Otherwise, the EJB method executes under either the RunAs identity or the caller identity, depending on the RunAs configuration. The J2EE RunAs specification is at the enterprise bean level. When RunAs identity is specified, it applies to all bean methods. The method level IBM RunAs extension introduced in Version 4.0 is still supported in this version.

Federal Information Processing Standards-approved

Federal Information Processing Standards (FIPS) are standards and guidelines issued by the National Institute of Standards and Technology (NIST) for federal computer systems. FIPS are developed when there are compelling federal government requirements for standards, such as for security and interoperability, but acceptable industry standards or solutions do not exist.

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which have undergone FIPS 140-2 certification. Throughout the documentation and the WebSphere Application Server, the IBM JSSE and JCE modules that have undergone FIPS certification are referred to as IBMJSSEFIPS and IBMJCEFIPS, which distinguishes the FIPS modules from the IBM JSSE and IBM JCE modules.

IBM i For more information, refer to Configuring Federal Information Processing Standard Java Secure Socket Extension files.

IBM i The IBMJSSEFIPS module supports the FIPS-approved Transport Layer Security (TLS) cipher suites including:

- SHA
- DES
- TripleDES

IBM i The IBMJSSEFIPS module supports the following algorithms:

- RSA public key algorithm
- ANSI X9.31

- IBM Random Number Generator (Patent pending)

The IBMJCEFIPS module supports the following symmetric cipher suites:
- AES (FIPS 197)
- TripleDES (FIPS 46-3)
- SHA1 Message Digest algorithm (FIPS 180-1)

The IBMJCEFIPS module supports the following algorithms:
- Digital Signature DSA and RSA algorithms (FIPS 186-2)
- ANSI X 9.31 (FIPS 186-2)
- IBM Random Number Generator

The IBMJCEFIPS cryptographic module contains the algorithms that are approved by FIPS, which form a proper subset of those in the IBM JCE modules.

What is new for security specialists

This version contains many new and changed features for those who are responsible for securing applications and the application serving environment.

- WebSphere Application Server security standards configurations

  WebSphere Application Server integrates cryptographic modules, which include Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE). Most of the requirements in the standards are handled in the JSSE and JCE, which must undergo the certification process to meet government standards. WebSphere Application Server must be configured to run with the JSSE and JCE enabled for a particular standard, and now supports the FIPS 140-2, SP800-131 and Suite B security standards.

What is new for securing web services

In WebSphere Application Server, there are many security enhancements for web services. The enhancements include supporting sections of the Web Services Security (WS-Security) specifications and providing architectural support for plugging in and extending the capabilities of security tokens.

Enhancements from the supported Web Services Security specifications

Since September 2002, the Organization for the Advancement of Structured Information Standards (OASIS) has been developing the Web Services Security (WS-Security) for SOAP message standard.

In April 2004, OASIS released the Web Services Security Version 1.0 specification, which is a major milestone for securing web services. In Feburary 2006, the specification was updated to Version 1.1. This specification is the foundation for other Web Services Security specifications and is also the basis for the Basic Security Profile (WS-I BSP) Version 1.0 specification, which was approved in March 2007.See the Basic Security Profile web page for more information.

Web Services Security Version 1.1 is a strategic move towards Web Services Security interoperability, and an important part of the Web Services Security roadmap. For more information on the Web Services Security roadmap, see Security in a Web Services World: A Proposed Architecture and Roadmap.

WebSphere Application Server supports the following OASIS specifications and WS-I profiles:
- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.1
- OASIS: Web Services Security: Kerberos Token Profile 1.1
- OASIS: WS-SecurityPolicy 1.2
- OASIS: WS-SecureConversation 1.3
- OASIS: WS-Trust 1.3

- Basic Security Profile (WS-I BSP) 1.0
- OASIS: Web Services Security: SAML Token Profile 1.1

The Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. Using SAML, a client can communicate assertions regarding the identity, attributes, and entitlements of a SOAP message. Using the SAML function in WebSphere Application Server, you can apply policy sets to JAX-WS applications to use SAML assertions in web services messages and in web services usage scenarios. Use SAML assertions to represent user identity and user security attributes, and optionally, to sign and to encrypt SOAP message elements.

For details on what parts of the previous specifications are supported in WebSphere Application Server, see "Supported functionality from OASIS specifications" on page 876.

High level features overview in WebSphere Application Server

In WebSphere Application Server, the Web Services Security for SOAP Message Version 1.1 specification is designed to be flexible and accommodate the requirements of Web services. For example, the specification does not have a mandatory security token definition. Instead, the specification defines a generic mechanism to associate the security token with a SOAP message. The use of security tokens is defined in the various Version 1.0 and 1.1 security token profiles, such as:

- The Username Token Profile
- The X.509 Token Profile
- The Kerberos Token Profile

For more information on security token profile development at OASIS, see Organization for the Advancement of Structured Information Standards.

The Web Services Security for SOAP Message Version 1.1 updates the Web Services Security for SOAP Message core specification and the various security token profiles. For this release, WebSphere Application Server implements the Username Token Profile 1.1 and the X.509 Token Profile 1.1, which includes support for the Thumbprint type of security token reference. In addition, it supports the signature confirmation and encrypted header portions of the Web Services Security Version 1.1 standard.

Important: The wire format (such as namespaces) in the WS-SecureConversation and WS-Trust 1.3 specification has changed. WebSphere Application Server tolerates requests formatted according to both the Submission Drafts and version 1.3 specifications, but you must ensure that the correct version is used when clients are communicating with a Web Services Feature Pack service provider. You can disable tolerance of the older format for WS-SecureConversation and WS-Trust 1.3 endpoints. Submission Drafts requests are not interoperable with version 1.3 standards.

WebSphere Application Server supports pluggable security tokens. The pluggable architecture is enhanced to support the Web Services Security specifications, other profiles, and other Web Services Security specifications. You can learn more about the pluggable security token framework for JAX-RPC web services, and associating custom security tokens with SOAP messages, by reading these articles on the IBM developerWorks website:

- Security for JAX-RPC Web services, Part 1: Generating custom tokens
- Security for JAX-RPC Web services, Part 2: Consuming custom tokens

WebSphere Application Server includes the following key enhancements:

- Support for the LTPA version 2 token
- Support for configuration of multiple callers, and an order attribute on the caller to determine which caller is used for the WebSphere credential
- Support for the published WS-SecurityPolicy version 1.2 specification embedded in WSDL

- Support for the WS-SecureConversation version 1.3 specification and the WS-Trust version 1.3 specification (used by WS-SecureConversation)
- Support for Kerberos token as defined in the WS-Kerberos Token Profile version 1.1 specification

For more information on some of these enhancements, see “Web Services Security enhancements” on page 872.

Configuration of Web Services Security

WebSphere Application Server uses the policy set model for implementing the Web Services Security Version 1.1 specification, including the Username token Version 1.1 profile, support for the Kerberos and LTPA v2 tokens, and the X.509 token version 1.1 profile. Policy sets combine configuration settings, including those for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, WS-SecureConversation, and WS-Security. For more information on policy sets, refer to the topic Managing policy sets using the administrative console.

You can use the administrative console to configure the Web Services Security binding of a deployed application with Web Services Security constraints that are defined in the policy set.

For the X.509 Certificate Token Profile, one new type of security token reference is the Thumbprint reference, which is specified in the binding. WebSphere Application Server now supports creating and authenticating a security token by using a security token reference (STR) with a key identifier and a Thumbprint in the <KeyInfo> element. The Thumbprint key information type requires that there be a keystore with the public and private key pair instead of a shared key. To use the Thumbprint of the specified certificate, specify the keyInfo type THUMBPRINT in the bindings.

For example, a decryption key is referenced by means of the thumbprint of an associated certificate. The certificate is not included in the message. Instead, the <ds:KeyInfo> element contains a <wsse:SecurityTokenReference> element that specified the thumbprint of the specified certificate by means of the http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1 attribute of the <wsse:KeyIdentifier> element.

To take advantage of implementations associated with the Web Services Security Version 1.1 specification, you must:

- Ensure that your applications use the Java API for XML Web Services (JAX-WS) programming model.
- Re-configure the Web Services Security constraints in the new policy set and binding format.

WebSphere Application Server provides the following tools that you can use to edit the policy set file and the binding file:

IBM assembly tools

You can use IBM assembly tools to develop web services and configure the policy set and the binding file for Web Services Security. The tools enable you to assemble both web and Enterprise JavaBeans (EJB) modules. The assembly tools do not support direct editing of policy sets, but can import policy sets from the application server, and then attach the modified policy sets to the service. For more information, read about assembly tools.

Note: You can use policy sets only with Java API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

WebSphere Application Server administrative console

You can use the administrative console to configure the Web Services Security binding of a deployed application with Web Services Security constraints that are defined in the policy set.

What is not supported

Web service security is still fairly new and some of the standards are still being defined or standardized. The following functionality is not supported in WebSphere Application Server:

- JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification). See the standard documentation for more information: JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification).
- Application programming interfaces (API) do not exist for Web Services Security in WebSphere Application Server Versions 6.0.x and later.
- SAML token profile is not supported out of the box.
- REL token profile is not supported.
- SwA profile is not supported

What is supported by the IBM Software Development Kit (SDK)

The following standards exist for the Java application programming interface for XML security and Web Services Security:

- JSR-105 (Java API for XML-Signature XPath Filter Version 2.0

  W3C Recommendation, November 2002
- JSR-106 (Java API for XML Encryption Syntax and Processing)

  W3C Recommendation, December 2002

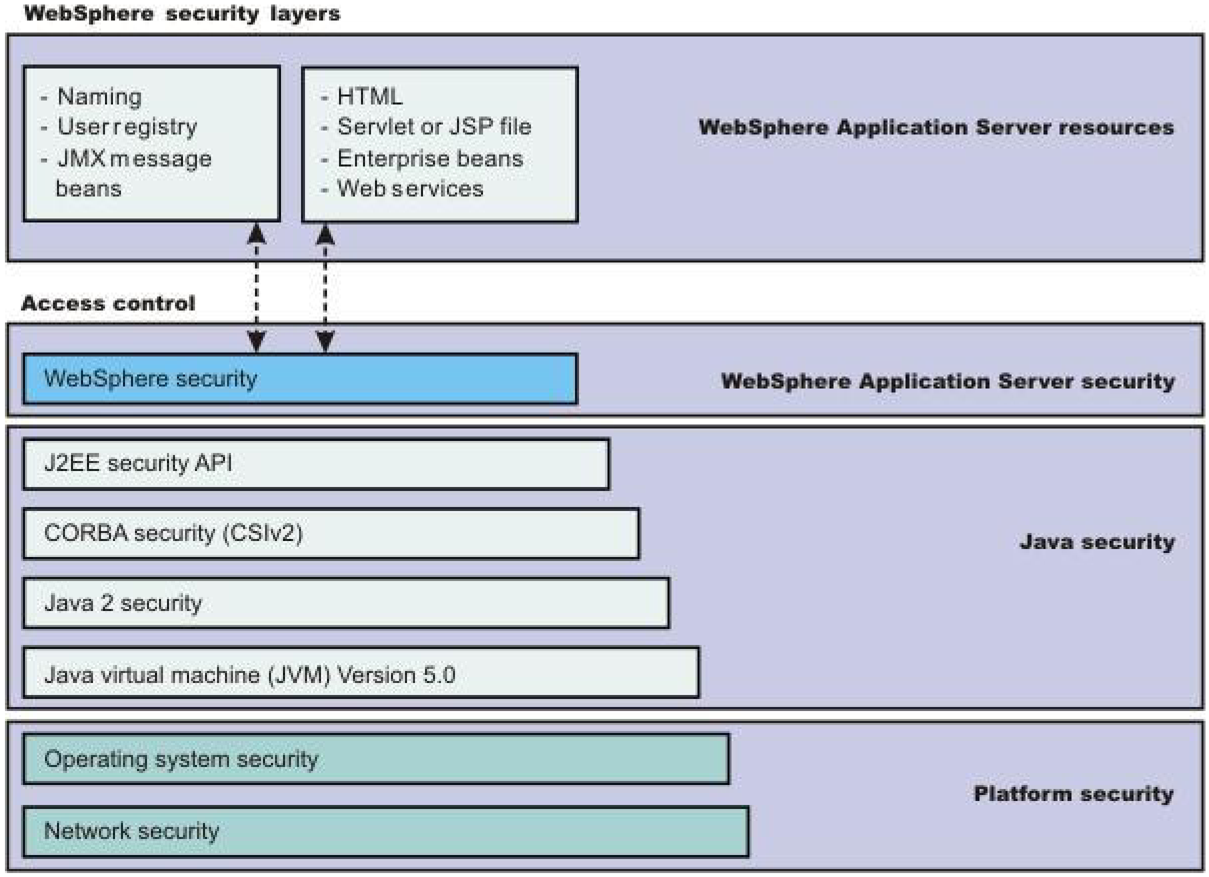
For more information on the IBM SDK for Java Version 6, see the security information documentation.

For information on what is supported for Web Services Security in WebSphere Application Server, see "Supported functionality from OASIS specifications" on page 876.

Security planning overview

When you access information on the Internet, you connect through web servers and product servers to the enterprise data at the back end. This section examines some typical configurations and common security practices.

This section also examines the security protection that is offered by each security layer and common security practice for good quality of protection in end-to-end security. The following figure illustrates the building blocks that comprise the operating environment for security within WebSphere Application Server:



The following information describes each of the components of WebSphere Application Server security, Java security, and Platform security that are illustrated in the previous figure.

WebSphere Application Server security

WebSphere security
WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

Java security

Java Platform, Enterprise Edition (Java EE) security application programming interface (API)
The security collaborator enforces Java Platform, Enterprise Edition (Java EE)-based security policies and supports Java EE security APIs.

Java 2 security
The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.

Java Virtual Machine (JVM) 5.0
The JVM security model provides a layer of security above the operating system layer. For example, JVM security protects the memory from unrestricted access, creates exceptions when errors occur within a thread, and defines array types.

Platform security

IBM i **Operating system security**

The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. These services include the file system security support that secures sensitive files in the product installation for WebSphere Application Server. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.

The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. The operating system identity of the servant, controller, and daemon Started Task, as established by the STARTED profile, is the identity that is used to control access to system resources such as files or sockets. Optionally, the operating system security can provide authentication services using the User Registry of local operating system, and/or authorization services using SAF Authorization for the WebSphere Administration console and for applications running under the application server.

In addition to knowledge of Secure Sockets Layer (SSL) and Transport Layer Security (TLS), the administrator must be familiar with System Authorization Facility (SAF) and Resource Access Control Facility (RACF), or an equivalent SAF based product.

The identity and verification of users can be managed by using a Local Operating System as the User Registry, RACF or equivalent SAF base product. Alternatively, an LDAP, Custom, or Federated User Registry can be used.

WebSphere can be configured to use SAF Authorization, which will use RACF or an equivalent SAF based product to manage and protect users and group resources. Alternatively, WebSphere can be configured to use WebSphere Authorization or a JACC External Authorization Provider.

When using either Local Operating System as the User Registry and/or using SAF Authorization, security auditing is an inherit feature of RACF or the equivalent SAF based products.

Network security

The Network Security layers provide transport level authentication and message integrity and confidentiality. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL). Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

Each product application server consists of a web container, an Enterprise Java Beans (EJB) container, and the administrative subsystem.

The administrative console is a special Java EE web application that provides the interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files, which must be protected by operating system security. Passwords and other sensitive configuration data can be modified using the administrative console. However, you must protect these passwords and sensitive data. For more information, see Encoding passwords in files.

The administrative console web application has a setup data constraint that requires access to the administrative console servlets and JavaServer Pages (JSP) files only through an SSL connection when administrative security is enabled.

IBM i In WebSphere Application Server Version 6.0.x and earlier, the administrator console HTTPS port was configured to use `DummyServerKeyFile.jks` and `DummyServerTrustFile.jks` with the default self-signed certificate. The dummy certificates and keys must be replaced immediately after WebSphere Application Server installation; the keys are common in all of the installation and are therefore insecure. WebSphere Application Server Version 6.1 provides integrated certificate and key management, which generate distinct private key and self-signed certificate with embedded server host name to enable host name verification. WebSphere Application Server Version 6.1 also enables integration with external certificate (CA) authority to use CA-issued certificates. The WebSphere Application Servers Version 6.1 installation process provides an option to enable administrative security during installation. As a result, a WebSphere Application Server process is secured immediately after installation. WebSphere Application Server Version 7.0 extends the embedded certificate management capabilities by creating a chained certificate (personal certificate signed by a root certificate) to enable refresh of the personal certificate without affecting the trust established. It also enables tailoring of the certificate during profile creation (you can import your own or change the distinguished name (DN) of the one created by default) as well as the ability to change the default keystore password.

Administrative security

IBM i WebSphere Application Servers interact with each other through CSIv2 and Secure Authentication Services (SAS) security protocols as well as the HTTP and HTTPS protocols.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

You can configure these protocols to use Secure Sockets Layer (SSL) when you enable WebSphere Application Server administrative security. The WebSphere Application Server administrative subsystem in every server uses SOAP, Java Management Extensions (JMX) connectors and Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) JMX connectors to pass administrative commands and configuration data. When administrative security is disabled, the SOAP JMX connector uses the HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When administrative security is enabled, the SOAP JMX connector always uses the HTTPS protocol. When administrative security is enabled, you can configure the RMI/IIOP JMX connector to either use SSL or to use TCP/IP. It is recommended that you enable administrative security and enable SSL to protect the sensitive configuration data.

Security for Java EE resources

Security for Java EE resources is provided by the web container and the EJB container. Each container provides two kinds of security: declarative security and programmatic security.

In declarative security, an application security structure includes network message integrity and confidentiality, authentication requirements, security roles, and access control. Access control is expressed in a form that is external to the application. In particular, the deployment descriptor is the primary vehicle for declarative security in the Java EE platform. WebSphere Application Server maintains Java EE security policy, including information that is derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At runtime, the container uses the security policy that is defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to express the security model of an application, you might use programmatic security to make access decisions. When administrative security is enabled and application server security is not disabled at the server level, Java EE applications security is enforced. When the security policy is specified for a web resource, the web container performs access control when the resource is requested by a web client. The web container challenges the web client for authentication data if none is present according to the specified authentication method, ensures that the data constraints are met, and determines whether the authenticated user has the required security role. The web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested servlet or JSP file if the user principal has one of the required security roles. Servlets and JSP files can use the HttpServletRequest methods, isUserInRole and getUserPrincipal.

IBM i When administrative security and application security are enabled, and the application server level application security is not disabled, the EJB container enforces access control on EJB method invocation.

The authentication occurs regardless of whether method permission is defined for the specific EJB method. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the EJBContext methods, isCallerInRole and getCallerPrincipal. Use the Java EE role-based access control to protect valuable business data from access by unauthorized users through the Internet and the intranet. Refer to Securing web applications using an assembly tool, and Securing enterprise bean applications.

Role-based security

WebSphere Application Server extends the security, role-based access control to administrative resources including the JMX system management subsystem, user registries, and Java Naming and Directory Interface (JNDI) name space. WebSphere administrative subsystem defines four administrative security roles:

Monitor role
A monitor can view the configuration information and status but cannot make any changes.

Operator role
An operator can trigger run-time state changes, such as start an application server or stop an application but cannot make configuration changes.

Configurator role
A configurator can modify the configuration information but cannot change the state of the runtime.

Administrator role
An operator as well as a configurator, which additionally can modify sensitive security configuration and security policy such as setting server IDs and passwords, enable or disable administrative security and Java 2 security, and map users and groups to the administrator role.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

WebSphere Application Server defines two additional roles that are available when you use wsadmin scripting only.

Deployer
A deployer can perform both configuration actions and run-time operations on applications.

Adminsecuritymanager
An administrative security manager can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups.

Auditor
An auditor can view and modify the configuration settings for the security auditing subsystem.

A user with the configurator role can perform most administrative work including installing new applications and application servers. Certain configuration tasks exist that a configurator does not have sufficient authority to do when administrative security is enabled, including modifying a WebSphere Application Server identity and password, Lightweight Third-Party Authentication (LTPA) password and keys, and assigning users to administrative security roles. Those sensitive configuration tasks require the administrative role because the server ID is mapped to the administrator role.

Enable WebSphere Application Server administrative security to protect administrative subsystem integrity. Application server security can be selectively disabled if no sensitive information is available to protect. For securing administrative security, refer to Authorizing access to administrative roles and Assigning users and groups to roles.

Java 2 security permissions

WebSphere Application Server uses the Java 2 security model to create a secure environment to run application code. Java 2 security provides a fine-grained and policy-based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. The Java EE Version 1.4 specification defines a typical set of Java 2 security permissions that web and EJB components expect to have.

Table 84. Java EE security permissions set for web components. The Java EE security permissions set for web components are shown in the following table.

Security Permission	Target	Action
java.lang.RuntimePermission	loadLibrary	
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.io.FilePermission	*	read, write
java.util.PropertyPermission	*	read

Table 85. Java EE security permissions set for EJB components. The Java EE security permissions set for EJB components are shown in the following table.

Security Permission	Target	Action
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.util.PropertyPermission	*	read

The WebSphere Application Server Java 2 security default policies are based on the Java EE Version 1.4 specification. The specification grants web components read and write file access permission to any file in the file system, which might be too broad. The WebSphere Application Server default policy gives web components read and write permission to the subdirectory and the subtree where the web module is installed. The default Java 2 security policies for all Java virtual machines and WebSphere Application Server processes are contained in the following policy files:

IBM i **`/QIBM/ProdData/Java400/jdk15/lib/security/java.policy`**
Used as the default policy for the Java virtual machine (JVM).

IBM i **`${USER_INSTALL_ROOT}/properties/server.policy`**
This file is used as the default policy for all product server processes.

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). The following files are the default policy files for a WebSphere Application Server subsystem. These policy files, which are an extension of the WebSphere Application Server runtime, are referred to as Service Provider Programming Interfaces (SPI), and shared by multiple Java EE applications:

IBM i

- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/spi.policy

  Used for embedded resources defined in the resources.xml file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.
- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/library.policy

  Used by the shared library that is defined by the WebSphere Application Server administrative console.
- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy

Used as the default policy for Java EE applications.

In general, applications do not require more permissions to run than those recommended by the Java EE specification to be portable among various application servers. However, some applications might require more permissions. WebSphere Application Server supports the packaging of a `was.policy` file with each application to grant extra permissions to that application.

Attention: Grant extra permissions to an application only after careful consideration because of the potential of compromising the system integrity.

Loading libraries into WebSphere Application Server does allow applications to leave the Java sandbox. WebSphere Application Server uses a permission filtering policy file to alert you when an application installation fails because of additional permission requirements. For example, it is recommended that you not give the java.lang.RuntimePermission exitVM permission to an application so that application code cannot terminate WebSphere Application Server.

The filtering policy is defined by the filtermask in the *profile_root*/`config/cells/`*`cell_name`*`/filter.policy` file. Moreover, WebSphere Application Server also performs run-time permission filtering that is based on the run-time filtering policy to ensure that application code is not granted a permission that is considered harmful to system integrity.

Therefore, many applications developed for prior releases of WebSphere Application Server might not be Java 2 security ready. To quickly migrate those applications to the latest version of WebSphere Application Server, you might temporarily give those applications the java.security.AllPermission permission in the `was.policy` file. Test those applications to ensure that they run in an environment where Java 2 security is active. For example, identify which extra permissions, if any, are required, and grant only those permissions to a particular application. Not granting the `AllPermission` permission to applications can reduce the risk of compromising system integrity. For more information on migrating applications, refer to Migrating Java 2 security policy.

The WebSphere Application Server runtime uses Java 2 security to protect sensitive run-time functions. Applications that are granted the `AllPermission` permission not only have access to sensitive system resources, but also WebSphere Application Server run-time resources and can potentially cause damage to both. In cases where an application can be trusted as safe, WebSphere Application Server does support having Java 2 security disabled on a per application server basis. You can enforce Java 2 security by default in the administrative console and clear the Java 2 security flag to disable it at the particular application server.

When you specify the `Enable administrative security` and Use Java 2 security to restrict application access to local resources options on the Global security panel of the administrative console, the information and other sensitive configuration data, are stored in a set of XML configuration files. Both role-based access control and Java 2 security permission-based access control are employed to protect the integrity of the configuration data. The example uses configuration data protection to illustrate how system integrity is maintained.

Attention: The `Enable global security` option in previous releases of WebSphere Application Server is the same as the `Enable administrative security` option in Version 8.5. Also, the Enable Java 2 security option in previous releases is the same as the Use Java 2 security to restrict application access to local resources option in Version 8.5.

- When Java 2 security is enforced, the application code cannot access the WebSphere Application Server run-time classes that manage the configuration data unless the code is granted the required WebSphere Application Server run-time permissions.
- When Java 2 security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless the code is granted the required file read and write permission.
- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and a RMI/IIOP remote interface to enable application programs to extract and to modify configuration files and data. When

administrative security is enabled, an application program can modify the WebSphere Application Server configuration if the application program has presented valid authentication data and the security identity has the required security roles.

- If a user can disable Java 2 security, the user can also modify the WebSphere Application Server configuration, including the WebSphere Application Server security identity and authentication data with other sensitive data. Only users with the administrator security role can disable Java 2 security.
- Because WebSphere Application Server security identity is given to the administrator role, only users with the administrator role can disable administrative security, change server IDs and passwords, and map users and groups to administrative roles, and so on.

IBM i

Other Runtime resources

Other WebSphere Application Server run-time resources are protected by a similar mechanism, as described previously. It is very important to enable WebSphere Application Server administrative security and to use Java 2 security to restrict application access to local resources. Java EE Specification defines several authentication methods for web components: HTTP Basic Authentication, Form-Based Authentication, and HTTPS Client Certificate Authentication. When you use client certificate login, it is more convenient for the browser client if the web resources have integral or confidential data constraint. If a browser uses HTTP to access the web resource, the web container automatically redirects the browser to the HTTPS port. The CSIv2 security protocol also supports client certificate authentication. You can also use SSL client authentication to set up secure communication among a selected set of servers based on a trust relationship.

If you start from the WebSphere Application Server plug-in at the web server, you can configure SSL mutual authentication between it and the WebSphere Application Server HTTPS server. When using a certificate, you can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Servers as shown in the following figure. Note that you can use self-signed certificates to reduce administration and cost.

For example, you want to restrict the HTTPS server in WebSphere Application Server A and in WebSphere Application Server B to accept secure socket connections only from the WebSphere Application Server plug-in W.

- IBM i To complete this task, you can generate three certificates using the IKEYMAN and the certificate management utilities. Also, you can use certificate W and trust certificate A and B. Configure the HTTPS server of WebSphere Application Server A to use certificate A and to trust certificate W.

Configure the HTTPS server of WebSphere Application Server B to use certificate B and to trust certificate W.

Table 86. Trust relationships from example. The trust relationship that is depicted in the previous figure is shown in the following table.

Server	Key	Trust
WebSphere Application Server plug-in	W	A, B
WebSphere Application Server A	A	W
WebSphere Application Server B	B	W

When WebSphere Application Server is configured to use Lightweight Directory Access Protocol (LDAP) user registry, you also can configure SSL with mutual authentication between every application server and the LDAP server with self-signed certificates so that a password is not visible when it is passed from WebSphere Application Server to the LDAP server.

IBM i WebSphere Application Server does not provide a registry configuration or management utility. In addition, it does not dictate the registry password policy. It is recommended that you use the password policy recommended by your registry, including the password length and expiration period.

Before securing your WebSphere Application Server environment, determine which versions of WebSphere Application Server you are using, review the WebSphere Application Server security architecture, and review each of the following topics:

- Common Secure Interoperability Version 2 features
- Identity assertion to the downstream server
- Selecting an authentication mechanism
  - IBM i Simple WebSphere authentication mechanism (deprecated)

    The Simple WebSphere Authentication Mechanism (SWAM) is deprecated in WebSphere Application Server Version 8.5. It will be removed in a future release. As an alternative, it is recommended that you use Lightweight Third Party Authentication (LTPA).
  - Lightweight Third Party Authentication
  - Trust associations
  - Single sign-on for authentication using LTPA cookies
- Selecting a registry or repository
  - Local operating system registries
  - Standalone Lightweight Directory Access Protocol registries
- Java 2 security
  - Java 2 security policy files
- Java Authentication and Authorization Service
  - Programmatic login for JAAS
- Java EE connector security
- Access control exception for Java 2 security
  - Role-based authorization
  - Administrative roles and naming service authorization
- Implementing a custom authentication provider using JASPI

Security considerations when registering a base Application Server node with the administrative agent

You might decide to centralize the control of your stand-alone base application servers by registering them with the administrative agent. If your base application server is currently configured with security, some issues require consideration. These security considerations apply to the use of the **registerNode** command and the **deregisterNode** command.

The goal of the **registerNode** command is to take a stand-alone base node and convert it into one that is managed by the administrative agent. The main parameter of the **registerNode** command is **profilePath**, which specifies where on the local machine the administrative agent can find the node. The **portsFile** parameter contains keys to determine which ports the administrative agent listens to, on behalf of the base node. The format is the same as that for **manageProfiles** command line.

The **registerNode** command is run from the administrative agent itself. It is used to register a node with an administrative agent. It is required that the administrative agent be on the same system as the node being registered. The **registerNode** command is only valid on a base node. If a node has been federated to a deployment manager the **registerNode** command fails with an error.

First, the exchange signers process for profile registration processes the default secure sockets layer (SSL) configuration, in which it obtains the root certificate signers from the NodeDefaultRootStore of the administrative agent and stores them in the NodeDefaultTrustStore of the target profile. Next, the process obtains the root certificate signers from the target profile's NodeDefaultRootStore and stores them in the

NodeDefaultTrustStore of the administrative agent. The signers are stored in the target profiles trust store using the alias prefix "agent_signer", and are stored in the administrative agents trust store using the alias prefix "<profileName>_signer".

Next, the exchange signers process for profile registration processes the RSAToken authentication configuration, in which it obtains the root certificate signers from the NodeRSATokenRootStore of the administrative agent and stores them in the NodeRSATokenTrustStore of the target profile. Next, the process obtains the root certificate signers from the target profile's NodeRSATokenRootStore and stores them in the NodeRSATokenTrustStore of the administrative agent. The signers are stored in the target profile's trust store using the alias prefix "agent_signer", and are stored in the administrative agents trust store using the alias prefix "<profileName>_signer".

In addition, the registration process stores all root certificate signers (SSL and RSAToken) from the administrative agent into the target profile's client trust store (ClientDefaultTrustStore by default).

The **deregisterNode** command activates the de-registration process which removes all signers exchanged during the registration process from both the administrative agent and base profile. The base node's configuration is retained, except that it is marked as not registered with an administrative agent. This command is only valid for a previously registered base node.

The following issues require consideration when running the **registerNode** command with security:

- When attempting to run system management commands such as the **registerNode** command, you need to explicitly specify administrative credentials to perform the operation. The **registerNode** command accepts -username and -password parameters to specify the user ID and password, respectively. The user ID and password that are specified must be for an administrative user; for example, a user that is a member of the console users with Administrator privileges or the administrative user ID configured in the user registry. An example of the **registerNode** command follows:

  ```
  registerNode -profilePath /WebSphere/AppServer/profiles/default -host localhost -connType SOAP -port 8877 -username WSADMIN -password ADMINPWD
  ```
- Before registering to an administrative agent, a node must have administrative security enabled or disabled.
- Once a node is registered, you cannot enable or disable the administrative security for that node (or any other registered node) until the node has been de-registered.

Proper understanding of the security interactions between distributed servers greatly reduces problems that are encountered with se cure communications. Security adds complexity because additional function needs management. The administrative agent provides a way of managing additional function while preserving security.

Security: Resources for learning

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

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

View links to additional information about:

- “Planning, business scenarios and IT architecture” on page 1019
- “Programming model and decisions” on page 1019
- “Programming specifications” on page 1019
- “Administration” on page 1019

- “Tutorials” on page 1020

Planning, business scenarios and IT architecture

- WebSphere Application Server Library
- WebSphere Application Server Support
- WebSphere Application Server Version 6 Security
- “Accessing the samples” on page 12

  The technology sample in the WebSphere Application Server Samples Gallery contains several security-related samples including the form login sample and the Java Authentication and Authorization Service (JAAS) login sample.
- WebSphere Application Server security: Presentation series
- IBM i AS/400® Tips and Tools for Securing your AS/400, SC41-5300

  This book provides a set of practical suggestions for using the security features and for establishing operating procedures that are security-conscious.
- IBM i OS/400® Security Reference, SC41-5302

  This book provides information about planning, setting up, managing, and auditing security. It describes all the features of security on the system and discusses how security features relate to other aspects of the system, such as work management, backup and recovery, and application design.

Programming model and decisions

- Federated Identity Management and Web Services Security with IBM Tivoli Security Solutions

Programming specifications

- J2EE Specifications
- EJB Specifications
- Servlet Specifications
- Common Secure Interoperability Version 2 (CSIv2) Specification
- IBM i JAAS Specification.

  For programming and usage in JAAS, refer to the specification located at http://www.ibm.com/developerworks/java/jdk/security/ and scroll down to find the JAAS documentation for your platform. This document contains the following when unpacked:
  - login.html - LoginModule Developer's Guide
  - api.html - Developer's Guide (JAAS JavaDoc)
  - HelloWorld.tar - Sample JAAS Application
- Java 2 Platform, Standard Edition, v5.0 API Specification
- Java Authorization Contract for Containers (JSR 115) Specification
- Java Authentication Service Provider Interface for Containers (JSR 196) Specification
- The Kerberos Network Authentication Service Version 5
- The Simple and Protected GSS-API Negotiation Mechanism
- Kerberos: The Network Authentication Protocol

Administration

- IBM i WebSphere Application Server Version 6: Security Handbook

  This is a redpiece or a draft version of WebSphere Application Server Version 6 Security handbook. It is designed to help programmers, administrators, and architects understand the features available in WebSphere Application Server Version 6.WebSphere Application Server V6 Migration Guide
- IBM i IBM WebSphere Version 5.0 Security

  This book provides an overview of WebSphere Application Server Version 5 Security, including J2EE security and programmatic security techniques. It also provides information about end-to-end security solutions that include WebSphere Application Server Version 5 as part of an e-business solution.
- IBM i AS/400 Internet Security: Protecting Your AS/400 from HARM in the Internet, SG24-4929-00

  This document describes what you need to know about security and how the different security elements fit together. This book explains the comprehensive security options that are available to secure your system and data.

- IBM i Domino and WebSphere Integration on the IBM eServer™ iSeries Server, SG24-6223-00

  This book documents the setup and configuration of an integrated environment with WebSphere Application Server and Lotus Domino. Part 1 of this redbook focuses on administration topics, such as single sign-on, the use of a common Lightweight Directory Access Protocol (LDAP) directory for authentication, and options for different HTTP stacks. Part 2 of this book focuses on investigating the application development topic of how WebSphere Application Server can access Lotus Domino applications. The book shows and discusses code snippets from example programs.
- IBM i HTTP Server (powered by Apache): An Integrated Solution for IBM iSeries Servers, SG24-6716-00

  This book is designed to help you plan, install, configure, troubleshoot, and understand the HTTP Server (powered by Apache) running on your server. The book explains how to configure the HTTP server for basic authentication, access control and SSL. The document also explains the steps to implement a web application using Java and WebSphere Application Server.
- IBM HTTP Server Support and Documentation
- IBM Directory Server Support and Documentation
- IBM developer kits

  This website provides access to the IBM developer kits that are provided by the IBM Centre for Java Technology Development. Using this website, you can find various security and diagnostic information including information on the Federal Information Processing Standard, Java Version 1.4.1, Java Version 1.4.2, the iKeyman tool, and the Public Key Cryptography Standards (PKCS).
- IBM cryptographic hardware devices
- Supported hardware, software and APIs prerequisite website
- IBM Education Assistant
- Understanding LDAP - Design and Implementation
- WebSphere security fundamentals
- IBM i
  - Digital certificate management

    Digital Certificate Manager (DCM) is a feature that centrally manages certificates for your applications.
  - IBM HTTP Server for iSeries

    Information for this topic applies to HTTP Server, which is powered by Apache.
  - Security and Directory Services

    Read this information to understand e-business security and Directory Services offerings.
- Advanced authentication in WebSphere Application Server
- IBM i WebSphere Application Server for iSeries V6: Building Advanced Configurations

Tutorials

- IBM Education Assistant: Enabling security best practices tutorials

  See these tutorials for overview information about WebSphere Application Server security.
- IBM i AS/400 Technical Studio: AS/400 Security Workshop

  The AS/400 Security Workshop covers topics such as Internet Security. Use your server Security Advisor to help determine your optimal security settings.

Chapter 31. Overview and new features for developing applications

View the topics in the following list to learn more about developing applications for deployment on this product.

What is new for developers

This topic provides an overview of new and changed features of the programming model and application serving environment as it pertains to development and test efforts.

Learn about WebSphere applications: Overview and new features

This topic provides an overview of the programming model.

Accessing the samples

The samples are a good way to become familiar with the programming model.

What is new for developers

This version contains many new and changed features for application developers.

- Configuring EJB bindings in SCA OASIS applications

  The SCA stateless session bean binding is a protocol binding that you can use to integrate SCA with EJB-based services. The product provides OASIS support for both EJB 3.x and 2.x bindings for services and references.
- Deploying OSGi applications

  The WebSphere Application Server administrative console provides panels that you can use to examine or debug a specific set of OSGi bundles running on an application server. You can, for example, view and explore the package and service dependencies between the bundles in an OSGi application.
- Developing asynchronous SCA services and clients

  SCA OASIS specifications support the asynchronous running of request-response services. This enables a client thread to continue doing other work while the service runs. You can use SCA OASIS annotations and APIs in Java interfaces to enable asynchronization in services.
- Developing OSGi applications
  - An OSGi application can contain Enterprise JavaBeans (EJBs). OSGi applications can access and invoke an enterprise bean directly.

    The enterprise beans in your OSGi bundles can be developed from scratch, or you can include existing EJB assets and migrate them to use OSGi modularity with minimal code changes. Stateful, stateless, and singleton enterprise beans are supported. Your OSGi application can also contain message-driven beans (MDBs).

    You deploy and configure an OSGi application that contains enterprise beans in a similar way to deploying and configuring a Java EE enterprise application, by using wsadmin or the administrative console.
  - You can configure bean security in the Blueprint XML file of your OSGi applications, so that the methods of the bean can be accessed only by users that are assigned a specified role. You can configure bean-level security, so that a single role is associated with all the methods of the bean, or you can configure method-level security, where different roles are associated with specific methods.
- Liberty profile: Developing applications

  The Liberty profile support in WebSphere Application Server delivers a new, lightweight profile of the application server for web and OSGi applications, along with a simplified configuration approach for developers. Key benefits of the Liberty profile include, but are not limited to, the following capabilities:
  - Free and frictionless download for development purposes
  - Ultra lightweight modular runtime environment, with an install size of under 50 MB

- Incredibly fast startup time: under 5 seconds for simple web applications
  - Simplified configuration for quick time to productivity
  - Java EE and OSGi application deployment support for web applications
  - LDAP registry support
  - Deployment, as a package, of an application and configured server
  - Managed, centralized deployment of a packaged application and server
  - Availability of WebSphere Application Server Developer Tools as Eclipse plug-ins for broad tooling support
  - Platform support for distributed platforms, z/OS, and Mac OS

  Incredibly fast restart times, coupled with its small size, dynamic behavior, and ease of use, make the Liberty profile a good option for developers building web applications that do not require the full Java EE environment of traditional enterprise application server profiles. Familiar WebSphereApplication Server enterprise qualities of service, such as security and transaction integrity, are enabled as required.
- SCA in WebSphere Application Server: Overview

  Version 8.5 adds support for the Service Component Architecture (SCA) OASIS programming model implementation.

What is new for deployers

This version contains many new and changed features for application deployers.

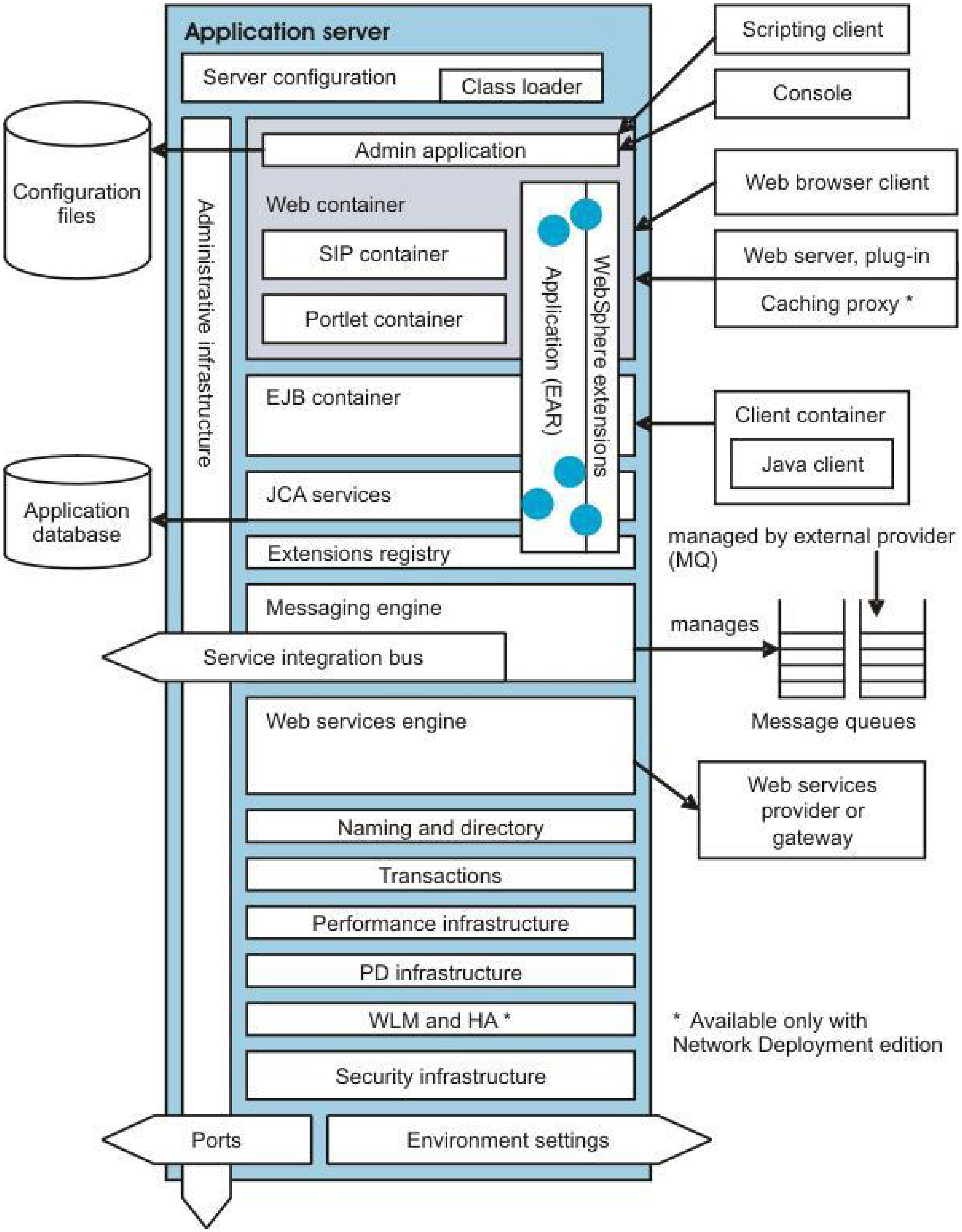
Learn about WebSphere applications: Overview and new features

Use the **Learn about WebSphere applications** section as a starting point to study the programing model, encompassing the many parts used in and by various application types supported by the application server.

The programming model for applications deployed on this product has the following aspects.

- Java specifications and other open standards for developing applications
- WebSphere programming model extensions to enhance application functionality
- Containers and services in the application server, used by deployed applications, and which sometimes can be extended

The diagram shows a single application server installation. The parts pertaining to the programming model are discussed here. Other parts comprise the product architecture, independent of the various application types outlined by the programming model. See “WebSphere Application Server architecture” on page 15.



Java EE application components

The product supports application components that conform to Java Platform, Enterprise Edition (Java EE) specifications.

Web applications run in the web container

The web container is the part of the application server in which web application components run. Web applications are comprised of one or more related servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files that you can manage as a unit. Combined, they perform a business logic function.

The web container processes servlets, JSP files, and other types of server-side includes. Each application server runtime has one logical web container, which can be modified, but not created or removed. Each web container provides the following.

Web container transport chains

Requests are directed to the web container using the web container inbound transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP requests, and a web container channel over which requests for servlets and JSP files are sent to the web container for processing.

Servlet processing

When handling servlets, the web container creates a request object and a response object, then invokes the servlet service method. The web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

Servlets can perform such tasks as supporting dynamic web page content, providing database access, serving multiple clients at one time, and filtering data.

JSP files enable the separation of the HTML code from the business logic in web pages. IBM extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming.

HTML and other static content processing

Requests for HTML and other static content that are directed to the web container are served by the web container inbound chain. However, in most cases, using an external web server and web server plug-in as a front end to the web container is more appropriate for a production environment.

Session management

Support is provided for the javax.servlet.http.HttpSession interface as described in the Servlet application programming interface (API) specification.

An *HTTP session* is a series of requests to a servlet, originating from the same user at the same browser. Sessions allow applications running in a web container to keep track of individual users. For example, many web applications allow users to dynamically collect data as they move through the site, based on a series of selections on pages they visit. Where the user goes next, or what the site displays next, might depend on what the user has chosen previously from the site. To maintain this data, the application stores it in a “session”.

SIP applications and their container

SIP applications are Java programs that use at least one Session Initiation Protocol (SIP) servlet. SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging.

Portlet applications and their container

Portlet applications are special reusable Java servlets that appear as defined regions on portal pages. Portlets provide access to many different applications, services, and web content.

EJB applications run in the EJB container

The EJB container provides all of the runtime services needed to deploy and manage enterprise beans. It is a server process that handles requests for both session and entity beans.

Enterprise beans are Java components that typically implement the business logic of Java EE applications, as well as accessing data. The enterprise beans, packaged in EJB modules, installed in an application server do not communicate directly with the server. Instead, the EJB container is an interface between EJB components and the application server. Together, the container and the server provide the enterprise bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative perspective, the container handles data access for the contained beans. A single container can host more than one EJB Java archive (JAR) file.

Client applications and other types of clients

In a client-server environment, clients communicate with applications running on the server. *Client applications* or *application clients* generally refers to clients implemented according to a particular set of Java specifications, and which run in the client container of a Java EE-compliant application server. Other clients in the WebSphere Application Server environment include clients implemented as web applications (*web clients*), clients of web services programs (*web services clients*), and clients of the product systems administration (*administrative clients*).

Client applications and their container

The client container is installed separately from the application server, on the client machine. It enables the client to run applications in an EJB-compatible Java EE environment. The diagram shows a Java client running in the client container.

This product provides a convenient launchClient tool for starting the application client, along with its client container runtime.

Depending on the source of technical information, client applications sometimes are called application clients. In this documentation, the two terms are synonymous.

Web clients, known also as web browser clients

The diagram shows a web browser client, which can be known simply as a web client, making a request to the web container of the application server. A web client or web browser client runs in a web browser, and typically is a web application.

Web services clients

Web services clients are yet another kind of client that might exist in your application serving environment. The diagram does not depict a web services client. The web services information includes information about this type of client.

Administrative clients

The diagram shows two kinds of administrative clients: a scripting client and the administrative console that is the graphical user interface (GUI) for administering this product. Both are accessing parts of the systems administration infrastructure. In the sense that they are basically the same for whatever kind of applications you are deploying on the server, administrative clients are part of the product architecture. However, because many of these clients are programs you create, they are discussed as part of the programming model for completeness.

Web services

Web services

The diagram shows the web services engine, part of the web services support in the application server runtime. Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a service-oriented architecture (SOA), which supports the connecting or sharing of resources and data in a flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

The product acts as both a web services provider and as a requestor. As a provider, it hosts web services that are published for use by clients. As a requester, it hosts applications that invoke web services from other locations. The diagram shows the web services engine in this capacity, contacting a web services provider or gateway.

SCA composites

Service Component Architecture (SCA)

SCA composites consist of components that implement business functions in the form of services.

Data access, messaging, and Java EE resources

Data access resources

Connection management for access to enterprise information systems (EIS) in the application server is based on the Java EE Connector Architecture (JCA) specification. The diagram shows JCA services helping an application to access a database in which the application retrieves and persists data.

The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts between the application server and EIS.

The Connection Manager (not shown) in the application server pools and manages connections. It is capable of managing connections obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions specification.

JDBC resources (JDBC providers and data sources) are a type of *Java EE resource* used by applications to access data. Although data access is a broader subject than that of JDBC resources, this information often groups data access under the heading of Java EE resources for simplicity.

JCA resource adapters are another type of Java EE resource used by applications. The JCA defines the standard architecture for connecting the Java EE platform to heterogeneous EIS. Imagine an ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between Java EE application servers or clients and an EIS. To use a resource adapter, install the resource adapter code and create configurations that use that adapter. The product provides a predefined relational resource adapter for your use.

Messaging resources and messaging engines

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Applications can use message-driven beans to automatically retrieve messages from JMS destinations and JCA endpoints without explicitly polling for messages.

For inbound non-JMS requests, message-driven beans use a Java EE Connector Architecture (JCA) 1.5 resource adapter written for that purpose. For JMS messaging, message-driven beans can use a JCA-based messaging provider such as the default messaging provider that is part of the product.

The messaging engine supports the following types of message providers.

Default messaging provider (service integration bus)

The default messaging provider uses the service integration bus for transport. The default message provider provides point-to-point functions, as well as publish and subscribe functions. Within this provider, you define JMS connection factories and destinations that correspond to service integration bus destinations.

WebSphere MQ provider
You can use WebSphere MQ as the external JMS provider. The application server provides the JMS client classes and administration interface, while WebSphere MQ provides the queue-based messaging system.

Generic JMS provider
You can use another messaging provider as long as it implements the ASF component of the JMS 1.0.2 specification. JMS resources for this provider cannot be configured using the administrative console.

transition: Version 6 replaces the Version 5 concept of a JMS server with a messaging engine built into the application server, offering the various kinds of providers mentioned previously. The Version 5 messaging provider is offered for configuring resources for use with Version 5 embedded messaging. You also can use the Version 5 default messaging provider with a service integration bus.

EJB 2.1 introduces an ActivationSpec for connecting message-driven beans to destinations. For compatibility with Version 5, you still can configure JMS message-driven beans (EJB 2.0) against a listener port. For those message-driven beans, the message listener service provides a listener manager that controls and monitors one or more JMS listeners, each of which monitors a JMS destination on behalf of a deployed message-driven bean.

Service integration bus

The service integration bus provides a unified communication infrastructure for messaging and service-oriented applications. The service integration bus is a JMS provider that provides reliable message transport and uses intermediary logic to adapt message flow intelligently into the network. It supports the attachment of web services requestors and providers. Its capabilities are fully integrated into product architecture, including the security, system administration, monitoring, and problem determination subsystems.

The service integration bus is often referred to as just a bus. When used to host JMS applications, it is often referred to as a messaging bus. It consists of the following parts (not shown at this level of detail in the diagram).

Bus members
Application servers added to the bus.

Messaging engine
The component that manages bus resources. It provides a connection point for clients to produce or from where to consume messages.

Destinations
The place within the bus to which applications attach to exchange messages. Destinations can represent web services endpoints, messaging point-to-point queues, or messaging publish and subscribe topics. Destinations are created on a bus and hosted on a messaging engine.

Message store
Each messaging engine uses a set of tables in a supported data store (such as a JDBC database) to hold information such as messages, subscription information, and transaction states.

Through the service integration bus web services enablement, you can:

- Make an internal service that is already available at a service destination available as a web service.
- Make an external web service available at a service destination.
- Use the web services gateway to map an existing service, either an internal service or an external web service, to a new web service that appears to be provided by the gateway.

Mail, URLs, and other Java EE resources

The following kinds of Java EE resources are used by applications deployed on a J2EE-compliant application server.

- JDBC resources and other technology for data access (previously discussed)
- JCA resource adapters (previously discussed)
- JMS resources and other messaging support (previously discussed)
- JavaMail support, for applications to send Internet mail

  The **JavaMail APIs** provide a platform and protocol-independent framework for building Java-based mail client applications. The APIs require service providers, known as protocol providers, to interact with mail servers that run on the appropriate protocols.

  A mail provider encapsulates a collection of protocol providers, including Simple Mail Transfer Protocol (SMTP) for sending mail; Post Office Protocol (POP) for receiving mail; and Internet Message Access Protocol (IMAP) as another option for receiving mail. To use another protocol, you must install the appropriate service provider for the protocol.

  JavaMail requires not only service providers, but also the JavaBeans Activation Framework (JAF), as the underlying framework to handle complex data types that are not plain text, such as Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.
- URLs, for describing logical locations

  URL providers implement the functionality for a particular URL protocol, such as HTTP, enabling communication between the application and a URL resource that is served by a particular protocol. A default URL provider is included for use by any URL resource with protocols based on the supported Java Platform, Standard Edition (Java SE) specification, such as HTTP, FTP, or File. You also can plug in your own URL providers that implement additional protocols.
- Resource environment entries, for mapping logical names to physical names

  The java:comp/env environment provides a single mechanism by which both the JNDI name space objects and local application environment objects can be looked up. The product provides numerous local environment entries by default.

  The Java EE specification also provides a mechanism for defining customer environment entries by defining entries in the standard deployment descriptor of an application. The Java EE specification uses the following methods to separate the definition of the resource environment entry from the application.

  – Requiring the application server to provide a mechanism for defining separate administrative objects that encapsulate a resource environment entry. The administrative objects are accessible using JNDI in the application server local name space (java:comp/env).

  – Specifying the administrative object's JNDI lookup name and expected returned object type. This specification is performed in the aforementioned resource environment entry in the deployment descriptor.

  The product supports the use of resource environment entries with the following administrative concepts.

  – A *resource environment entry* defines the binding target (JNDI name), factory class, and return object type (via the link to a referenceable) of the resource environment entry.

  – A *referenceable* defines the class name of the factory that returns object instances implementing a Java interface.

  – A *resource environment provider* groups together the referenceable, resource environment entries and any required custom properties.

Security

Security programming model and infrastructure

The product provides security infrastructure and mechanisms to protect sensitive Java EE resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

Security infrastructure and mechanisms protect Java Platform, Enterprise Edition (Java EE) resources and administrative resources, addressing your enterprise security requirements. In turn, the security infrastructure of this product works with the existing security infrastructure of your multiple-tier enterprise computing framework. Based on open architecture, the product provides many plug-in points to integrate with enterprise software components to provide end-to-end security.

The security infrastructure involves both a programming model and elements of the product architecture that are independent of the application type.

Additional services for use by applications

Naming and directory

Each application server provides a naming service that in turn provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register resources hosted on the application server. The JNDI implementation is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model used by application developers. CosNaming provides the server-side implementation and is where its name space is actually stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming, and interacts with the CosNaming server on behalf of the client.

Clients of the application server use the naming architecture to obtain references to objects related to those applications. The objects are bound into a mostly hierarchical structure called the name space. It consists of a set of name bindings, each one of which is a name relative to a specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.

This product provides the following naming and directory features.

- Distributed name space, for additional scalability
- Transient and persistent partitions, for binding at various scopes
- Federated name space structure across multiple servers
- Configured bindings for defining bindings bound by the system at server startup
- Support for CORBA Interoperable Naming Service (INS) object URLs

Note that with the addition of virtual member manager to provide federated repository support for product security, the product now offers more extensive and sophisticated identity management capabilities than ever before, especially in combination with other WebSphere and Tivoli products.

Object Request Broker (ORB)

The product uses an ORB to manage interaction between client applications and server applications, as well as among product components. An ORB uses IIOP to enable clients to make requests and receive requests from servers in a network distributed environment.

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency.

Although not shown in the diagram, one place in which the ORB comes into play is where the client container is contacting the EJB container on behalf of a Java client.

Transactions

Part of the application server is the transaction service. The product provides advanced transactional capabilities to help application developers avoid custom coding. It provides support for the many challenges related to integrating existing software assets with a Java EE environment. These measures include ActivitySessions (described below).

Applications running on the server can use transactions to coordinate multiple updates to resources as one unit of work such that all or none of the updates are made permanent. Transactions are started and ended by applications or the container in which the applications are deployed.

The application server is a transaction manager that supports coordination of resource managers and participates in distributed global transactions with other compliant transaction managers.

The server can be configured to interact with databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction support is not required.

How applications use transactions depends on the type of application, for example:

- A session bean either can manage its transactions itself, or delegate the management of transactions to the container.
- Entity beans use container-managed transactions.
- Web components, such as servlets, use bean-managed transactions.

The product handles transactions with the following components.

- A transaction manager supports the enlistment of recoverable XAResources and ensures each resource is driven to a consistent outcome, either at the end of a transaction, or after a failure and restart of the application server.
- A container manages the enlistment of XAResources on behalf of deployed applications when it performs updates to transactional resource managers such as databases. Optionally, the container can control the demarcation of transactions for EJB applications that have enterprise beans configured for container-managed transactions.
- An API handles bean-managed enterprise beans and servlets, allowing such application components to control the demarcation of their own transactions.

WebSphere extensions

WebSphere programming model extensions are the programming model benefits you gain by purchasing this product. They represent leading edge technology to enhance application capability and performance, and make programming and deployment faster and more productive.

In addition, your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis. The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers who build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. For more information about this feature, see Application extension registry.

The various WebSphere programming model extensions, and the corresponding application services that support them in the application server runtime, can be considered in three groups: Business Object Model extensions, Business Process Model extensions, and extensions for producing Next Generation Applications.

Extensions pertaining to the Business Object Model

Business object model extensions operate with business objects, such as enterprise bean (EJB) applications.

Application profiling

Application profiling is a WebSphere extension for defining strategies to dynamically control concurrency, prefetch, and read-ahead.

Application profiling and access intent provide a flexible method to fine-tune application performance for enterprise beans without impacting source code. Different enterprise beans, and even different methods in one enterprise bean, can have their own intent to access resources. Profiling the components based on their access intent increases performance in the application server runtime.

Dynamic query

Dynamic query is a WebSphere programming extension for unprecedented application flexibility. It lets you dynamically build and submit queries that select, sort, join, and perform calculations on application data at runtime. Dynamic Query service provides the ability to pass in and process EJB query language queries at runtime, eliminating the need to hard-code required queries into deployment descriptors during application development.

Dynamic query improves enterprise beans by enabling the client to run custom queries on EJB components during runtime. Until now, EJB lookups and field mappings were implemented at development time and required further development or reassembly in order to be changed.

Dynamic cache

The dynamic cache service improves performance by caching the output of servlets, commands, and JSP files. This service within the application server intercepts calls to cacheable objects and either stores the output of the object or serves the content of the object from the dynamic cache.

Because Java EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create opportunity for significant gains in server response time, throughput, and scalability.

Features include cache replication among clusters, cache disk offload, Edge side include caching, and external caching - the ability to control caches outside of the application server, such as that of your Web server.

Extensions pertaining to the Business Process Model

Business process model extensions provide process, workflow functionality, and services for the application server. Use them in conjunction with business integration capabilities.

ActivitySessions

ActivitySessions are a WebSphere extension for reducing the complexity of dealing with commitment rules and limitations associated with one-phase commit resources.

ActivitySessions provide the ability to extend the scope of multiple local transactions, and to group them. This enables them to be committed based on deployment criteria or through explicit program logic.

Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Extensions for creating next generation applications

Next generation extentions can be used in applications that need the specific extensions. These enable next generation development by leveraging the latest innovations that build on today's Java EE standards. This provides greater control over application development, execution, and performance than was ever possible before.

Asynchronous beans

Asynchronous beans offer performance enhancements for resource-intensive tasks by enabling single tasks to run as multiple tasks. Asynchronous scheduling facilities can also be used to process parallel processing requests in “batch mode” at a designated time. The product provides full support for asynchronous execution and invocation of threads and components within the application server. The application server provides execution and security context for the components, making them an integral part of the application.

Startup beans

Startup beans allow the automatic execution of business logic when the application server starts or stops. For example, they might be used to pre-fill application-specific caches, initialize application-level connection pools, or perform other application-specific initialization and termination procedures.

Object pools

Object pools provide an effective means of improving application performance at runtime, by allowing multiple instances of objects to be reused. This reuse reduces the overhead associated with instantiating, initializing, and garbage-collecting the objects. Creating an object pool allows an application to obtain an instance of a Java object and return the instance to the pool when it has finished using it.

Internationalization

The internationalization service is a WebSphere extension for improving developer productivity. It allows you to automatically recognize the time zone and location information of the calling client, so that your application can act appropriately. The technology enables you to deliver each user, around the world, the right date and time information, the appropriate currencies and languages, and the correct date and decimal formats.

Scheduler

The scheduler service is a WebSphere programming extension responsible for starting actions at specific times or intervals. It helps minimize IT costs and increase application speed and responsiveness by maximizing utilization of existing computing resources. The scheduler service provides the ability to process workloads using parallel processing, set specific transactions as high priority, and schedule less time-sensitive tasks to process during low traffic off-hours.

Work areas

Work areas are a WebSphere extension for improving developer productivity. Work areas provide a capability much like that of “global variables”. They provide a solution for passing and propagating contextual information between application components.

Work areas enable efficient sharing of information across a distributed application. For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it will be available throughout your application, eliminating the need to hand-code a solution or to read and write information to a database.

Specifications and API documentation

The WebSphere Application Server product supports various industry standards. This topic lists the specifications and application programming interface (API) documentation supported in current and previous product releases.

Components

- Any application type
- Web applications
- Portlet applications

- SIP applications
- EJB applications
- OSGi applications
- Client applications
- Web services
- Service integration
- Data access resources
- Messaging resources
- Mail, URLs, and other Java EE resources
- Security
- Web Services Security
- Naming and directory
- Object Request Broker
- Transactions
- WebSphere extensions
- Administration

The **Version 8.5** column in the tables lists the latest specification level that the product supports. However, support for specifications is compatible with earlier versions of the product; the Version 8.5 product supports all specifications that are listed for Version 6.0 through Version 8.5. For example, for any application type, the Version 8.0 product supports Java EE 5 and 6 and J2EE 1.2, 1.3, and 1.4. The word "New" beside a specification indicates that the product first supported the specification in that product version.

Any application type

Table 87. Supported specifications for any application type. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Platform, Enterprise Edition (Java EE) specification Prior to Java EE 5, the specification name was Java 2 Platform, Enterprise Edition (J2EE).	Java EE 6 (JSR 316)	Java EE 6 (JSR 316) New	Java EE 5 New	J2EE 1.4	J2EE 1.4 New J2EE 1.3 J2EE 1.2
Java Platform, Standard Edition (Java SE) specification Prior to Java SE 6, the specification name was Java 2 Platform, Standard Edition (J2SE).	Java SE 7 New	Java SE 6	Java SE 6 New	J2SE 5	J2SE 1.4.2
ISO 8859 specifications	ISO 8859 applies to these versions.				

Web applications

Table 88. Supported specifications for web applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Servlet specification (JSR 154, JSR 53 and JSR 315)	Java Servlet 3.0	Java Servlet 3.0 New	Java Servlet 2.5 New	Java Servlet 2.4	Java Servlet 2.4 New Java Servlet 2.3

Table 88. Supported specifications for web applications (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
JavaServer Faces (JSF) specification (JSR 252 and 127)	Apache MyFaces - JSF 2.0	Apache MyFaces - JSF 2.0 New	Sun Reference Implementation - JSF 1.2 Apache MyFaces 1.2 - JSF 1.2	JSF 1.1	JSF 1.0
JavaServer Pages (JSP) specification (JSR 245, JSR 152, and JSR 53)	JSP 2.2	JSP 2.2 New	JSP 2.1 New	JSP 2.0	JSP 2.0 New JSP 1.2

Portlet applications

Table 89. Supported specifications for portlet applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Portlet specification	Portlet 2.0 (JSR 286)	Portlet 2.0 (JSR 286)	Portlet 2.0 (JSR 286) New	Portlet 1.0 (JSR 168)	Not applicable. The product supports portlets in Version 6.1.

Session Initialization Protocol applications

Table 90. Supported specifications and APIs for SIP applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Session Initiation Protocol (SIP) Servlet API For a complete list of SIP and SIP proxy standards, see “SIP industry standards compliance” on page 642.	SIP 1.1 (JSR 289)	SIP 1.1 (JSR 289) New	SIP 1.1 (JSR 289) New for Feature Pack for CEA 1.0	SIP 1.0 (JSR 116)	Not applicable. The product supports SIP in Version 6.1.

Enterprise bean (EJB) applications

Table 91. Supported specifications and APIs for EJB applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Enterprise JavaBeans (EJB) specification	EJB 3.1	EJB 3.1 New	EJB 3.0	EJB 3.0 New for Feature Pack for EJB 3.0	EJB 2.1 New EJB 2.0 EJB 1.1

Table 91. Supported specifications and APIs for EJB applications (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New JDBC 2.1 and Optional Package API (2.0)
Java Message Service (JMS) specification	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1 New
Java Persistence API (JPA) specification	JPA 2.0	JPA 2.0	JPA 2.0 New for Feature Pack for OSGi and JPA 2.0	JPA 1.0 New for Feature Pack for EJB 3.0	Not applicable

OSGi applications

Table 92. Supported specifications and APIs for OSGi applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
OSGi Service Platform specification	OSGi Service Platform Release 4 Version 4.2	OSGi Service Platform Release 4 Version 4.2	OSGi Service Platform Release 4 Version 4.2 New for Feature Pack for OSGi and JPA 2.0	Not applicable	Not applicable
OSGi Alliance RFC-0112 Bundle Repository specification	OSGi Alliance RFC-0112 (Draft)	OSGi Alliance RFC-0112 (Draft)	OSGi Alliance RFC-0112 (Draft) New for Feature Pack for OSGi and JPA 2.0	Not applicable	Not applicable

Client applications

Table 93. Supported specifications and APIs for client applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Web Start architecture	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2 New

Web services

Table 94. Supported specifications and APIs for web services. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Architecture for XML Binding (JAXB) specification	JAXB 2.2	JAXB 2.2 New	JAXB 2.1 New	JAXB 2.0 New for Feature Pack for Web Services	Not applicable
Java Architecture for XML Binding (JAXB) Reference Implementation Vendor Extensions Runtime Properties specification	JAXB 2.2 RI Vendor Extensions	JAXB 2.2 RI Vendor Extensions New	JAXB 2.1 RI Vendor Extensions New	JAXB 2.0 RI Vendor Extensions New for Feature Pack for Web Services	Not applicable
Java API for XML Processing (JAXP) specification	1.4 Included in Java SE 6.	1.4 Included in Java SE 6.	1.4 Included in Java SE 6.	1.3 Included in J2SE 5.	1.2 Maintenance release of JSR 63
Java API for XML Registries (JAXR) specification	JAXR 1.0	JAXR 1.0	JAXR 1.0	JAXR 1.0	JAXR 1.0 New
Java API for XML-based RPC (JAX-RPC) specification	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1 New
Java API for RESTful Web Services (JAX-RS) specification	JAX-RS 1.1	JAX-RS 1.1 New			
Java API for XML Web Services (JAX-WS) specification	JAX-WS 2.2	JAX-WS 2.2 New	JAX-WS 2.1 New	JAX-WS 2.0 New for Feature Pack for Web Services	Not applicable
SOAP	SOAP 1.2	SOAP 1.2	SOAP 1.2	SOAP 1.2 New for Feature Pack for Web Services	SOAP 1.1
SOAP with Attachments API for Java (SAAJ) Specification	SAAJ 1.3	SAAJ 1.3	SAAJ 1.3	SAAJ 1.3 New for Feature Pack for Web Services	SAAJ 1.2 New
SOAP over Java Message Service (SOAP over JMS)	W3C SOAP over JMS 1.0	W3C SOAP over JMS 1.0	W3C SOAP over JMS 1.0 (submission draft)		
SOAP Message Transmission Optimization Mechanism (MTOM)	MTOM 1.0	MTOM 1.0	MTOM 1.0	MTOM 1.0 New for Feature Pack for Web Services	Not applicable
Streaming API for XML (StAX)	StAX 1.0	StAX 1.0	StAX 1.0	StAX 1.0 New for Feature Pack for Web Services	Not applicable
Universal Description, Discovery and Integration (UDDI)	UDDI 3.0	UDDI 3.0	UDDI 3.0	UDDI 3.0	UDDI 3.0 New

Table 94. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
W3C XML Schema	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2
Web Services Addressing (WS-Addressing) For more information, see "Web Services Addressing version interoperability" on page 787.	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • Core • SOAP Binding • WSDL Binding • WS-Addressing WSDL Binding, W3C Last Call • WS-Addressing, W3C Submission	Not applicable
Web Services Atomic Transaction (WS-AT)	WS-AT 1.2	WS-AT 1.2	WS-AT 1.1 New WS-AT 1.2 New	WS-AT 1.0	WS-AT 1.0 New
Web Services Business Activity (WS-BA)	WS-BA 1.2	WS-BA 1.2	WS-BA 1.1 New WS-BA 1.2 New	WS-BA 1.0	Not applicable
Web Services Coordination (WS-COOR)	WS-COOR 1.2	WS-COOR 1.2	WS-COOR 1.1 New WS-COOR 1.2 New	WS-COOR 1.0	WS-COOR 1.0 New
Web Services Description Language (WSDL)	WSDL 1.1	WSDL 1.1	WSDL 1.1	WSDL 1.1	WSDL 1.1
Web Services for Java Platform, Enterprise Edition (Java EE) (JSR 109) Prior to Web Services for Java EE, the specification name was Web Services for Java 2 Platform, Enterprise Edition (J2EE).	JSR 109 1.3	JSR 109 1.3 New	JSR 109 1.2 New	JSR 109 1.1	JSR 109 1.1 New

Table 94. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	**Version 8.5**	**Version 8.0**	**Version 7.0**	**Version 6.1**	**Version 6.0**
Web Services Interoperability Organization (WS-I) Basic Profile	WS-I Basic Profile 1.2 WS-I Basic Profile 2.0	WS-I Basic Profile 1.2 WS-I Basic Profile 2.0	WS-I Basic Profile 1.2 (draft) WS-I Basic Profile 2.0 (draft)	WS-I Basic Profile 1.2 (draft) New for Feature Pack for Web Services WS-I Basic Profile 2.0 (draft) New for Feature Pack for Web Services	WS-I Basic Profile 1.1 New
Web Services-Interoperability (WS-I) Attachments Profile	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0 New
Web Services Interoperability (WS-I) Reliable Secure Profile (RSP) Prior to WS-I RSP, the specification was named Reliable Asynchronous Messaging Profile (RAMP)	WS-I RSP 1.0	WS-I RSP 1.0	RAMP 1.0	RAMP 1.0 New for Feature Pack for Web Services	Not applicable
Web Services Invocation Framework (WSIF)	WSIF	WSIF	WSIF	WSIF	WSIF
Web Services Metadata for the Java Platform (JSR 181)	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform New for Feature Pack for Web Services	Not applicable
Web Services Notification (WS-Notification)	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	Not applicable

Table 94. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Web Services Policy (WS-Policy) specification	Web Services Policy 1.5 Web Services Addressing 1.0 - Metadata Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 WS-SecurityPolicy 1.2	Web Services Policy 1.5 Web Services Addressing 1.0 - Metadata Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 WS-SecurityPolicy 1.2	Web Services Policy 1.5 New Web Services Addressing 1.0 - Metadata New Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 New Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 New WS-SecurityPolicy 1.2 New	Not applicable	
Web Services Reliable Messaging	WS-MakeConnection Version 1.0	WS-MakeConnection Version 1.0	WS-MakeConnection Version 1.0 New	WS-ReliableMessaging 1.0 and WS-ReliableMessaging 1.1. New for Feature Pack for Web Services	Not applicable
Web Services Resource Framework (WSRF)	WSRF 1.2	WSRF 1.2	WSRF 1.2	WSRF 1.2 New	Not applicable
XML-binary Optimized Packaging (XOP)	XOP 1.0	XOP 1.0	XOP 1.0	XOP 1.0 New for Feature Pack for Web Services	Not applicable

Service integration

Table 95. Supported specifications and APIs for service integration. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New

Data access resources

Table 96. Supported specifications and APIs for data access resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0
Service Data Objects (SDO) specification				SDO 1.0	SDO 1.0 New

Messaging resources

Table 97. Supported specifications and APIs for messaging resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Message Service (JMS)	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0

Mail, URLs, and other Java EE resources

Table 98. Supported specifications and APIs for mail, URLs, and other Java EE resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
JavaMail API documentation (JSR 919)	JavaMail 1.4	JavaMail 1.4	JavaMail 1.4 New	JavaMail 1.3	JavaMail 1.3 New
URL API documentation	URL 1.4.2	URL 1.4.2	URL 1.4.2	URL 1.4.2	URL 1.4.2 New
JavaBeans Activation Framework (JAF) Specification	JAF 1.1	JAF 1.1	JAF 1.1 New	JAF 1.0.2	JAF 1.0.2 New
W3C Architecture - Naming and Addressing: URIs, URLs	W3C Naming and Addressing applies to these versions.				

Security

Table 99. Supported specifications and APIs for security. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java 2 Security Manager	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.4 New

Table 99. Supported specifications and APIs for security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Authentication and Authorization Service (JAAS)	JAAS 2.0 applies to these versions.				
Java Authorization Contract for Containers (JACC)	JACC 1.4	JACC 1.4	JACC 1.4 New	JACC 1.0	JACC 1.0 New
Java Authentication Service Provider Interface for Containers (JASPI)	JASPI 1.0	JASPI 1.0	Not applicable	Not applicable	Not applicable
Common Secure Interoperability Version 2 (CSIv2) specification This is an Object Management Group (OMG) CORBA/IIOP specification.	CSI 2.0 applies to these versions.				
Secure Sockets Layer (SSL) configuration The product uses Java Secure Sockets Extension (JSSE) as the SSL implementation for secure connections. JSSE is part of the Java 2 Standard Edition (J2SE) specification and is included in the IBM implementation of the Java Runtime Extension (JRE) specification.	JSSE 5.0	JSSE 5.0	JSSE 5.0	JSSE 5.0 New	JSSE 1.0.3
Java Generic Security Service (JGSS) Use JGSS with the Kerberos Network Authentication Service, Version 5	JGSS 1.0.1 applies to these versions.				
The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)	SPNEGO 1.0 applies to these versions.				
Java Cryptographic Extension (JCE) specification	JCE 1.0 applies to these versions.				
Java Certification Path (CertPath) API	CertPath 1.1	CertPath 1.1	CertPath 1.1	CertPath 1.1 New	CertPath 1.0

Web Services Security

Table 100. Supported specifications and APIs for Web Services Security. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Canonical XML	Canonical XML 1.0 applies to these versions.				
Decryption Transform for XML Signature	Decryption Transformation for XML Signature applies to these versions. .				
Exclusive XML Canonicalization	Exclusive XML Canonicalization 1.0 applies to these versions.				
OASIS Web Services Security: SOAP Message Security (WS-Security)	WS-Security 1.1	WS-Security 1.1	WS-Security 1.1	WS-Security 1.1 New for Feature Pack for Web Services	WS-Security 1.0

Table 100. Supported specifications and APIs for Web Services Security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
OASIS Web Services Security: Kerberos Token Profile	Kerberos Token Profile 1.1	Kerberos Token Profile 1.1	Kerberos Token Profile 1.1 New	Not applicable	
OASIS Web Services Security: SAML Token Profile 1.1 **Note:** WebSphere Application Server supports this specification in reference to the SAML Version 1.1 and 2.0 assertions within SOAP messages only.	SAML Version 1.1 and 2.0 assertions	SAML Version 1.1 and 2.0 assertions	SAML Version 1.1 and 2.0 assertions		
OASIS Web Services Security: Username Token Profile	Username Token Profile 1.1	Username Token Profile 1.1	Username Token Profile 1.1	Username Token Profile 1.1 New for Feature Pack for Web Services	Username Token Profile 1.0 New
OASIS Web Services Security: X.509 Token Profile	X.509 Token Profile 1.1	X.509 Token Profile 1.1	X.509 Token Profile 1.1	X.509 Token Profile 1.1 New for Feature Pack for Web Services	X.509 Token Profile 1.0 New
Web Services Interoperability Organization (WS-I) Basic Security Profile	WS-I Basic Security Profile 1.1	WS-I Basic Security Profile 1.1	WS-I Basic Security Profile 1.1 New	WS-I Basic Security Profile 1.0	Not applicable
Web Services Interoperability Organization (WS-I) Reliable Secure Profile	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft) New for Feature Pack for Web Services	Not applicable
Web Services Secure Conversation (WS-SecureConversation)	OASIS WS-SecureConversation 1.3	OASIS WS-SecureConversation 1.3	OASIS WS-SecureConversation 1.3 New	OASIS WS-SecureConversation 1.0 (draft submission) New for Feature Pack for Web Services	Not applicable

Table 100. Supported specifications and APIs for Web Services Security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Web Services Trust	OASIS WS-Trust 1.3	OASIS WS-Trust 1.3	OASIS WS-Trust 1.3 New	OASIS WS-Trust 1.1 (draft) New for Feature Pack for Web Services	Not applicable
XML Signature Syntax and Processing	XML Signature Syntax and Processing applies to these versions.				
XML Encryption Syntax and Processing	XML Encryption Syntax and Processing applies to these versions.				

Naming and directory

Table 101. Supported specifications and APIs for naming and directory. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Naming and Directory Interface (JNDI) Specification See also “JNDI support in WebSphere Application Server” on page 467.	JNDI on Java SE 7 New	JNDI on Java SE 6	JNDI on Java SE 6 New	JNDI on J2SE applies to these versions.	
Common Object Request Broker: Architecture and Specification (CORBA) specification This is an Object Management Group (OMG) Interoperable Naming (CosNaming) specification.	CORBA 2.4 applies to these versions.				
Interoperable Naming Service specification This is an OMG CosNaming specification.	Interoperable Naming Service				
Naming Service specification This is an OMG CosNaming specification.	Naming Service applies to these versions.				

Object Request Broker

The Object Request Broker (ORB) component follows the Common Object Request Broker Architecture (CORBA) specifications supported by Java 2 Platform, Standard Edition (J2SE). The Object Management Group (OMG) produces the specifications.

Versions 6.1 and later use the J2SE 5.0 specifications that are listed in *Official Specifications for CORBA support in J2SE 5.0* at http://download.oracle.com/javase/1.5.0/docs/guide/idl/compliance.html.

Version 6.0.x uses the J2SE 1.4 specifications that are listed in *Official Specifications for CORBA support in J2SE 1.4* at http://download.oracle.com/javase/1.4.2/docs/api/org/omg/CORBA/doc-files/compliance.html.

Table 102. Supported specifications and APIs for ORB. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Common Object Request Broker Architecture (CORBA) specifications	CORBA 2.3.1 applies to these versions.				
Revised IDL to Java language mapping	Revised IDL to Java language mapping applies to these versions.				
New IDL to Java Mapping Chapter	New IDL to Java Mapping Chapter applies to these versions.				
Updated Java to IDL Mapping specification	Updated Java to IDL Mapping applies to these versions.				
Interoperable Naming Service revised chapters	Interoperable Naming Service revised chapters applies to these versions.				
Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification New	Not applicable
Portable Interceptors specification	Not applicable	Not applicable	Not applicable	Not applicable	Portable Interceptors specification

Transactions

Table 103. Supported specifications and APIs for transactions. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
CORBA Object Transaction Service (OTS) specification	OTS 1.4	OTS 1.4	OTS 1.4	OTS 1.4	OTS 1.4 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0
Java Transaction API (JTA) specification	JTA 1.1	JTA 1.1	JTA 1.1 New	JTA 1.0.1B	JTA 1.0.1B New
Java Transaction Service (JTS) specification	JTS 1.0 applies to these versions.				

WebSphere extensions

Table 104. Supported specifications and APIs for WebSphere extensions. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
ActivitySession service and Last Participant Support					
J2EE Activity Service for Extended Transactions (JSR 95)	JSR 95 applies to these versions.				
Java Transaction API (JTA) specification	JTA 1.1	JTA 1.1	JTA 1.1 New	JTA 1.0.1B New	JTA 1.0.1
Internationalization (i18n)					

Table 104. Supported specifications and APIs for WebSphere extensions (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
J2SE internationalization documentation	J2SE Internationalization 5.0	J2SE Internationalization 5.0	J2SE Internationalization 5.0	J2SE Internationalization 5.0 New	J2SE Internationalization 1.4.2

Administration

Table 105. Supported specifications and APIs for administration. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java EE Application Deployment specification	Java EE Deployment 1.2	Java EE Deployment 1.2	Java EE Deployment 1.2 New	J2EE Deployment 1.1	J2EE Deployment 1.1 New
J2EE Extension Mechanism Architecture	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2 New
Java Management Extensions (JMX) JSR-000003	JMX 1.4	JMX 1.4	JMX 1.4	JMX 1.2	JMX 1.2 New
Java Management Extensions (JMX) Remote API	JMX Remote API 1.0	JMX Remote API 1.0	JMX Remote API 1.0	JMX Remote API 1.0 New	Not applicable
Java Virtual Machine (JVM) specification See WebSphere Application Server detailed system requirements.	JVM 6	JVM 6	JVM 6 New	JVM 5.0 New	JVM 1.4.2
Logging API specification (JSR 47)	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0 New

Introduction: Web services

Explore the key concepts pertaining to web services applications. Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a service-oriented architecture (SOA), which supports the connecting or sharing of resources and data in a flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

The WebSphere® Application Server supports a set of web services standards that support the creation and administration of interoperable, securable, transactionable, and reliable web services applications. Using the strategic Java™ API for XML-Based Web Services (JAX-WS) programming model, web service clients can now additionally invoke web services asynchronously, which means your client can continue processing without waiting on the response. Your JAX-WS web services can also take advantage of the Web Services Reliable Messaging protocol quality of service where you can be confident that your communication is reliable and reaches its destination while interoperating with other vendors.

The WebSphere® Application Server supports both the JAX-WS programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation web services programming model extending the foundation provided by the JAX-RPC programming model.

JAX-WS simplifies development through a standard, annotation-based model to develop web services and clients. A common set of binding rules for XML and Java objects make it easy to incorporate XML data and processing functions in Java. A further set of enhancements help you optimally send binary attachments, such as images or files with the web services request.

Simplified management of these web services profiles makes it easy to configure and reuse configurations, so you can seamlessly incorporate new web services profiles.

For a complete list of the supported standards and specifications, see the web services specifications and API documentation.

Introduction: Messaging resources

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider)
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider)
- Third-party messaging providers that implement either a JCA Version 1.5 or 1.6 resource adapter or the ASF component of the JMS Version 1.0.2 specification

Your applications can use messaging resources from any of these JMS providers. The choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you might already have a messaging infrastructure based on WebSphere MQ. In this case, you can either connect directly by using the WebSphere MQ messaging provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is a logical choice. If your business also uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominately WebSphere MQ network, choose the WebSphere MQ messaging provider. To administer a third-party messaging provider, you use either the resource adaptor (for a Java EE Connector Architecture (JCA) 1.5-compliant or 1.6-compliant messaging provider) or the client (for a non-JCA messaging provider) that is supplied by the third party.

Introduction: Dynamic cache

Explore the key concepts pertaining to the dynamic cache service, which improves performance by caching the output of servlets, commands, web services, and JavaServer Pages (JSP) files.

The dynamic cache engine is the default cache provider for the dynamic cache APIs and framework. However, starting with Version 6.1.0.27, dynamic cache allows WebSphere eXtreme Scale, which is the is the strategic direction for caching for the WebSphere products, to act as its core caching engine.

Configuring the dynamic cache to use WebSphere eXtreme Scale lets you leverage transactional support, improved scalability, high availability, and other WebSphere eXtreme Scale features without changing your existing dynamic cache caching code. If you are currently using the default cache provider, you can use the administrative console or wsadmin commands to replace the default dynamic cache provider with the WebSphere eXtreme Scale dynamic cache provider. You do not have to make any changes to your dynamic cache programming model. See the topic WebSphere eXtreme Scale dynamic cache provider in the WebSphere eXtreme Scale Version 7.0 Information Center for an overview of the WebSphere eXtreme Scale dynamic cache provider.

WebSphere eXtreme Scale can operate as an in-memory database processing space. You can use this processing space to provide in-line caching for a database back-end, or as a side cache. in-line caching uses WebSphere eXtreme Scale as the primary means for interacting with the data. When WebSphere eXtreme Scale is used as a side cache, the back-end is used in conjunction with WebSphere eXtreme Scale.

Functional advantages of using the WebSphere eXtreme Scale dynamic cache provider

The WebSphere eXtreme Scale dynamic cache provider:

- Supports memory-to-memory replication for sessions.
- Can handle all of the generic data and session caching needs of your applications.
- Enables your applications to leverage system memory without using SAN or storage solutions to host a dynamic cache disk cache.
- Provides a scalable replicated cache with a configurable number of replicas, thereby eliminating the need to use the data replication service (DRS), which the default cache provider uses. Use of DRS sometimes causes performance problems.
- Can be configured with additional WebSphere eXtreme Scale containers at runtime, thereby increasing your cache capacity, and preventing performance issues that sometimes occur when you use DRS. WebSphere eXtreme Scale automatically redistributes the partitions as new containers are added to the grid.
- Provides better caching qualities of service and control, than the default cache provider.
- Uses the same runtime monitoring and administration tools as the classic dynamic cache. These tools, such as the cache monitor and the dynamic cache runtime MBean, work the same way when dynamic cache runs on top of WebSphere eXtreme Scale, as when they are used with classic dynamic cache.

Functional differences between the default cache provider and the WebSphere eXtreme Scale dynamic cache provider

Following is a list of functional differences between the default cache provider and the WebSphere eXtreme Scale dynamic cache provider:

- WebSphere eXtreme Scale dynamic cache provider does not include disk cache support because all cache data is kept in memory. Therefore, the disk caching custom properties are not supported.
- WebSphere eXtreme Scale dynamic cache provider does not support the following features:
  - DistributedNioMap - skipMemoryAndWriteToDisk
  - DistributedMap and DistributedNioMap alias
  - Disabling dependency IDs or templates
  -
- WebSphere eXtreme Scale dynamic cache provider does not support DRS style replication. Therefore, the DRS custom properties are not supported.

  When you use the WebSphere eXtreme Scale dynamic cache provider, replication configuration is controlled by the WebSphere eXtreme Scale deployment and definition files. See the topic Configuring the dynamic cache provider for WebSphere eXtreme Scale in the WebSphere eXtreme Scale Version 7 Information Center for more information about replication.

- DistributedNioMapObject.release() is not called to release the byteBuffers for NIO buffer Management.
- WebSphere eXtreme Scale dynamic cache provider has limited PMI support. Certain PMI and MBean counters are no longer valid. See the topic Configuring the dynamic cache provider for WebSphere eXtreme Scale in the WebSphere eXtreme Scale Version 7 Information Center for more information.
- When firing any event, ObjectGrid always sets the sourceOfInvalidation to `REMOTE`
- You can use the DynaCache API to register event listeners regardless of which cache provider you are using. However, if you use WebSphere eXtreme Scale dynamic cache provider, the event listeners work as expected for local in-memory caches. When co-located containers are being used, events are thrown on the machine where the request that caused the event is serviced instead of on the machine where the request originated. For example if an invalidate request is issued on Server A and the cache entry that gets invalidated is actually stored on Server B, then the event will be fired on Server B. With the default dynamic cache provider, the event is fired on Server A. When stand-alone containers are being used, no events are fired through the DynaCache event listener API.

Key concepts pertaining to the dynamic cache service

Explore the key concepts pertaining to the dynamic cache service, which improves performance by caching the output of servlets, commands, web services, and JavaServer Pages (JSP) files.

Cache instances
An application uses a cache instance to store, retrieve, and share data objects within the dynamic cache.

Using the dynamic cache service to improve performance
Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance. WebSphere Application Server consolidates several caching activities including servlets, web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server.

Configuring dynamic cache to use the WebSphere eXtreme Scale dynamic cache provider
Configuring the dynamic cache service to use WebSphere eXtreme Scale lets you leverage transactional support, improved scalability, high availability, and other WebSphere eXtreme Scale features without changing your existing dynamic cache caching code.

Configuring servlet caching
After a servlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the servlet. These side effects can include calls to other servlets or JavaServer Pages (JSP) files or metadata about the entry, including timeout and entry priority information.

Configuring portlet fragment caching
After a portlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the portlet. These side effects can include calls to other portlets or metadata about the entry, including timeout and entry priority information.

Eviction policies using the disk cache garbage collector
The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

Configuring the JAX-RPC web services client cache
The web services client cache is a part of the dynamic cache service that is used to increase the performance of web services clients by caching responses from remote web services.

Cache monitor
Cache monitor is an installable web application that provides a real-time view of the current state

of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a GUI interface to manually change data.

Invalidation listeners

Invalidation listener mechanism uses Java events for alerting applications when contents are removed from the cache.

Learn about SIP applications

Find links to web resources for learning, including conceptual overviews, tutorials, samples, and “How do I?...” topics.

How do I?...

Configure SIP applications
Configuring the SIP container
SIP timer summary
SIP container settings
Session Initiation Protocol (SIP) container inbound channel settings
Session Initiation Protocol (SIP) inbound channel settings
IBM i Performing controlled failover of SIP applications

Develop SIP applications
Developing SIP applications
“SIP industry standards compliance” on page 642
“Runtime considerations for SIP application developers” on page 644
Developing a custom trust association interceptor
Developing SIP applications that support PRACK
“SIP IBM Rational Application Developer for WebSphere framework” on page 645
Setting up SIP application composition
“SIP servlets” on page 646
“SIP SipServletRequest and SipServletResponse classes” on page 647
“SIP SipSession and SipApplicationSession classes” on page 647
“Example: SIP servlet simple proxy” on page 648
“Example: SIP servlet SendOnServlet class” on page 650
“Example: SIP servlet Proxy servlet class” on page 651

Deploy SIP applications
Deploying SIP applications
Deploying SIP applications through the console
Deploying SIP applications through scripting

Secure SIP applications
Securing SIP applications
Configuring security for the SIP container

Trace SIP applications
Tracing a SIP container

Monitor SIP applications
SIP PMI counters

Troubleshoot SIP applications
Troubleshooting SIP applications

Conceptual overviews

Using Session Initiation Protocol to provide multimedia and interactive services
"SIP in WebSphere Application Server" on page 641
"SIP applications" on page 642
"SIP container" on page 655

developerWorks articles

Introducing SIP
Developing converged applications

Tutorials

Tutorials are not available at this time.

Samples

Samples are not available at this time.

Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

Your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application.

Introduction: Dynamic cache

Explore the key concepts pertaining to the dynamic cache service, which improves performance by caching the output of servlets, commands, web services, and JavaServer Pages (JSP) files.

The dynamic cache engine is the default cache provider for the dynamic cache APIs and framework. However, starting with Version 6.1.0.27, dynamic cache allows WebSphere eXtreme Scale, which is the is the strategic direction for caching for the WebSphere products, to act as its core caching engine.

Configuring the dynamic cache to use WebSphere eXtreme Scale lets you leverage transactional support, improved scalability, high availability, and other WebSphere eXtreme Scale features without changing your existing dynamic cache caching code. If you are currently using the default cache provider, you can use the administrative console or wsadmin commands to replace the default dynamic cache provider with the WebSphere eXtreme Scale dynamic cache provider. You do not have to make any changes to your dynamic cache programming model. See the topic WebSphere eXtreme Scale dynamic cache provider in the WebSphere eXtreme Scale Version 7.0 Information Center for an overview of the WebSphere eXtreme Scale dynamic cache provider.

WebSphere eXtreme Scale can operate as an in-memory database processing space. You can use this processing space to provide in-line caching for a database back-end, or as a side cache. in-line caching uses WebSphere eXtreme Scale as the primary means for interacting with the data. When WebSphere eXtreme Scale is used as a side cache, the back-end is used in conjunction with WebSphere eXtreme Scale.

Functional advantages of using the WebSphere eXtreme Scale dynamic cache provider

The WebSphere eXtreme Scale dynamic cache provider:

- Supports memory-to-memory replication for sessions.
- Can handle all of the generic data and session caching needs of your applications.
- Enables your applications to leverage system memory without using SAN or storage solutions to host a dynamic cache disk cache.
- Provides a scalable replicated cache with a configurable number of replicas, thereby eliminating the need to use the data replication service (DRS), which the default cache provider uses. Use of DRS sometimes causes performance problems.
- Can be configured with additional WebSphere eXtreme Scale containers at runtime, thereby increasing your cache capacity, and preventing performance issues that sometimes occur when you use DRS. WebSphere eXtreme Scale automatically redistributes the partitions as new containers are added to the grid.
- Provides better caching qualities of service and control, than the default cache provider.
- Uses the same runtime monitoring and administration tools as the classic dynamic cache. These tools, such as the cache monitor and the dynamic cache runtime MBean, work the same way when dynamic cache runs on top of WebSphere eXtreme Scale, as when they are used with classic dynamic cache.

Functional differences between the default cache provider and the WebSphere eXtreme Scale dynamic cache provider

Following is a list of functional differences between the default cache provider and the WebSphere eXtreme Scale dynamic cache provider:

- WebSphere eXtreme Scale dynamic cache provider does not include disk cache support because all cache data is kept in memory. Therefore, the disk caching custom properties are not supported.
- WebSphere eXtreme Scale dynamic cache provider does not support the following features:
  - DistributedNioMap - skipMemoryAndWriteToDisk
  - DistributedMap and DistributedNioMap alias
  - Disabling dependency IDs or templates
  -
- WebSphere eXtreme Scale dynamic cache provider does not support DRS style replication. Therefore, the DRS custom properties are not supported.

  When you use the WebSphere eXtreme Scale dynamic cache provider, replication configuration is controlled by the WebSphere eXtreme Scale deployment and definition files. See the topic Configuring the dynamic cache provider for WebSphere eXtreme Scale in the WebSphere eXtreme Scale Version 7 Information Center for more information about replication.
- DistributedNioMapObject.release() is not called to release the byteBuffers for NIO buffer Management.

- WebSphere eXtreme Scale dynamic cache provider has limited PMI support. Certain PMI and MBean counters are no longer valid. See the topic Configuring the dynamic cache provider for WebSphere eXtreme Scale in the WebSphere eXtreme Scale Version 7 Information Center for more information.
- When firing any event, ObjectGrid always sets the sourceOfInvalidation to `REMOTE`
- You can use the DynaCache API to register event listeners regardless of which cache provider you are using. However, if you use WebSphere eXtreme Scale dynamic cache provider, the event listeners work as expected for local in-memory caches. When co-located containers are being used, events are thrown on the machine where the request that caused the event is serviced instead of on the machine where the request originated. For example if an invalidate request is issued on Server A and the cache entry that gets invalidated is actually stored on Server B, then the event will be fired on Server B. With the default dynamic cache provider, the event is fired on Server A. When stand-alone containers are being used, no events are fired through the DynaCache event listener API.

Key concepts pertaining to the dynamic cache service

Explore the key concepts pertaining to the dynamic cache service, which improves performance by caching the output of servlets, commands, web services, and JavaServer Pages (JSP) files.

Cache instances
An application uses a cache instance to store, retrieve, and share data objects within the dynamic cache.

Using the dynamic cache service to improve performance
Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance. WebSphere Application Server consolidates several caching activities including servlets, web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server.

Configuring dynamic cache to use the WebSphere eXtreme Scale dynamic cache provider
Configuring the dynamic cache service to use WebSphere eXtreme Scale lets you leverage transactional support, improved scalability, high availability, and other WebSphere eXtreme Scale features without changing your existing dynamic cache caching code.

Configuring servlet caching
After a servlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the servlet. These side effects can include calls to other servlets or JavaServer Pages (JSP) files or metadata about the entry, including timeout and entry priority information.

Configuring portlet fragment caching
After a portlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the portlet. These side effects can include calls to other portlets or metadata about the entry, including timeout and entry priority information.

Eviction policies using the disk cache garbage collector
The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

Configuring the JAX-RPC web services client cache
The web services client cache is a part of the dynamic cache service that is used to increase the performance of web services clients by caching responses from remote web services.

Cache monitor
Cache monitor is an installable web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a GUI interface to manually change data.

Invalidation listeners

Invalidation listener mechanism uses Java events for alerting applications when contents are removed from the cache.

Accessing the samples

The product offers samples that demonstrate common enterprise application tasks. Many samples also provide instructions for deployment and coding examples.

The product provides samples in two ways:

Plants By WebSphere sample installed with the product

If you select to install samples when installing the product and when creating an application server profile, the Plants By WebSphere application is included with the product. The application demonstrates several Java Platform, Enterprise Edition (Java EE) functions, using an online store that specializes in plant and garden tool sales.

See Installing the Plants By WebSphere sample.

Samples downloadable from the Samples, Version 8.5 information center

The product provides component-specific samples that you can download at any time from a download site.

- Available samples
- Downloading samples

Installing the Plants By WebSphere sample

To install the Plants By WebSphere sample, perform the following steps.

1. Install the product.

   Plants By WebSphere sample is installed in the *app_server_root*/`samples` directory. A Plants By WebSphere pre-built enterprise archive named `pbw-ear.ear` is in the `/samples/PlantsByWebSphere/pbw-ear/target` directory.

   Installation instructions are in the `/samples/PlantsByWebSphere/docs` directory.

   You can build or modify the sample source code to support your project. The source code is in a `src` directory.
2. Start the application server.

Available samples

Samples that you can download include, for example, the following materials:

Communications Enabled Applications (CEA) samples

The CEA sample applications provide two main services, telephony access and multi-modal web interaction. Use this collection of sample applications to explore the services and to use as a starting point when developing your own communication enabled applications.

OSGi samples

The OSGi samples help you develop and deploy modular applications that use both Java EE and OSGi technologies.

XML samples

The XML samples demonstrate use of the XML API and supported specifications.

Internationalization service sample

The Internationalization service sample demonstrates how to use the internationalization service in Java EE applications, specifically within servlets and enterprise beans.

Web services samples

These samples demonstrate both Java API for XML-based RPC (JAX-RPC) and Java API for XML Web Services (JAX-WS) web services that use Java Platform, Enterprise Edition (Java EE) beans and JavaBeans components.

The JAX-WS web service samples demonstrate the implementation of one-way and two-way web services that highlight the use of web services standards such as WS-Addressing (WS-A) , WS-Reliable Messaging (WS-RM), and WS-Secure Conversation (WS-SC) and the SOAP Message Transmission Optimization Mechanism (MTOM) technology.

Service Data Objects (SDO) sample

This sample demonstrates data access to a relational database through Service Data Objects (SDO) and Java DataBase Connectivity (JDBC) Mediator technologies.

Downloading samples

You can download samples from the **Samples, Version 8.5** information center.

1. Go to the **Samples, Version 8.5** information center.
2. Determine which samples you want to download.
3. On the **Downloads** tab for the samples that you want, click a **Download Sample** link.
4. In the authentication window, click **OK**.
5. Download the compressed file, or individual sample files, to a directory on your workstation.

   You might create the `/samples/`*`sample_type`* directory path on your workstation and download the sample files to that directory path.

Many sample compressed files have an `/installableApps` directory that contains deployable prebuilt archives. Other directories contain files such as sample-specific source archives, scripts, and instructions for building deployable archives.

Limitations of the samples

- The samples are for demonstration purposes only.

  The code that is provided is not intended to run in a secured production environment. The samples support Java 2 Security, therefore the samples implement policy-based access control that checks for permissions on protected system resources, such as file I/O.

  The samples also support administrative security.

Additional samples and examples

Samples on developerWorks

Additional product samples are available on WebSphere developerWorks

Samples in tutorials

Many product tutorials rely on sample code. To find tutorials that demonstrate specific technologies, browse the links in “Tutorials” on page 12.

Examples in the product documentation

The product documentation contains many code snippets and examples. To locate these examples easily, see the developer examples in the **Reference** section of the information center navigation for the product edition that you are using.

IBM i **IBM Telephone Directory**

The IBM Telephone Directory business application is shipped separately from the product. For information about obtaining and using the IBM Telephone Directory application, see IBM Telephone Directory V5.2 in the topic on e-business and web serving in the IBM i information center.

Mail, URLs, and other J2EE resources

This topic describes the supported resources that are defined by Java Platform, Enterprise Edition (Java EE).

The product supports all of the resources defined by the Java EE. It adds the following resources in support of service extensions:

- Schedulers
- Work managers
- Object pools

Data access (JDBC and J2C)

The J2EE Connector architecture defines a standard architecture that enables the integration of various enterprise information systems (EIS) with application servers and enterprise applications. It defines a standard resource adapter used by a Java application to connect to an EIS. This resource adapter can plug into the application server and, through the Common Client Interface (CCI), provide connectivity between the EIS, the application server, and the enterprise application.

For more information, refer to “Data access resources” on page 997.

Messaging

The product supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The base JMS support enables the product applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

For more information, refer to “Messaging resources” on page 998.

Mail

Using JavaMail API, a code segment can be embedded in any Java EE application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

For more information, refer to JavaMail API.

URLs

Java EE applications can use URLs as resources in the same way other Java EE resources, such as JDBC and JavaMail, are used.

For more information, refer to “URLs” on page 342.

Resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

For more information, see Configuring new resource environment entries to map logical environment resource names to physical names.

Data access resources

These topics provide information about accessing data resources.

The connection between an enterprise application and an enterprise information system (EIS) is accomplished through the use of EIS-provided resource adapters, which are plugged into the application server. The resource adapter plays a central role in the integration and connectivity between an EIS and an application server. It serves as the point of contact between application components, application servers, and enterprise information systems. A resource adapter must communicate with other components based on well-defined contracts that are specified by the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA).

Generic inflow context enables a resource adapter to control the execution context of the Work instances it submits to the application server. By submitting a Work instance that implements the WorkContextProvider interface, a resource adapter can provide various types of context to the WebSphere Application Server. If the application server supports the provided context types, the generic work context mechanism sets the work contexts as the execution context of the Work instance. The context remains effective during the execution the Work instance.

Security inflow context uses generic work context by enabling a resource adapter to establish security information in the execution context of the Work instances that it submits to the application server. By submitting a Work instance that provides context types by implementing the new standardized SecurityContext interface, the application can establish and set an execution context containing the security identities and credentials for a Work instance. The identities and credentials remain effective during the execution of the Work instance, ensuring secure message delivery to Java EE message endpoints.

WebSphere Application Server supports work context types that implement the new standardized SecurityContext, TransactionContext and HintsContext interfaces. The generic inflow context mechanism accepts implementations of the HintsContext interface, but the application server does not act upon these implementations of the HintsContext interface. The security inflow context mechanism does not map user identities from the EIS domain to identities in an application server domain. Identities provided by implementations of SecurityContext must be in a security domain of application server.

Consult the following concept, reference, and task files for more overview information.

Messaging resources

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages. Applications can use point-to-point and publish/subscribe messaging. These styles of messaging can be used in the following ways: one-way; request and response; one-way and forward.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages. Message-driven beans can be configured as listeners on a Java EE Connector Architecture (JCA) 1.5 or 1.6 resource adapter or against a listener port.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider)
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider)
- Third-party messaging providers that implement either a JCA Version 1.5 or 1.6 resource adapter or the ASF component of the JMS Version 1.0.2 specification

Your applications can use messaging resources from any of these JMS providers. The choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you might already have a messaging infrastructure based on WebSphere MQ. In this case, you can either connect directly by using the WebSphere MQ messaging provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is a logical choice. If your business also uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominately WebSphere MQ network, choose the WebSphere MQ messaging provider. To administer a third-party messaging provider, you use either the resource adaptor (for a Java EE Connector Architecture (JCA) 1.5-compliant or 1.6-compliant messaging provider) or the client (for a non-JCA messaging provider) that is supplied by the third party.

For more information, see “Introduction: Messaging resources” on page 1046.

Chapter 32. Overview and new features for monitoring

Use the links provided in this topic to learn about monitoring capabilities.

New for administrators: Improved monitoring and performance tuning
A section of this topic describes what is new in the area of performance tuning.

Performance: Resources for learning

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

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas. The following sections are covered in this reference:

View the following links for additional information:

- "Request metrics "
- "Monitoring performance with third-party tools"
- "Tuning performance"
- "Java™ performance resource" on page 1060
- IBM i "Performance resources" on page 1060

Request metrics

- Systems Management: Application Response Measurement (ARM)

  The Open Group ARM specifications.

Monitoring performance with third-party tools

- IBM Search Solutions.

  Use IBM's Global Solution Directory to find a list of IBM's business partners that offer performance monitoring tools compliant with WebSphere Application Server.

Tuning performance

- IBM i WebSphere and Java tuning tips

  This page provides links to papers and articles that can help you take advantage of the latest WebSphere Application Server for IBM i performance improvements, tools, and tuning methods to optimize WebSphere Application Server performance.
- IBM i Tuning the WebSphere Prepared Statement Cache

  This PDF manual describes how to configure the prepared statement cache in WebSphere Application Server for IBM i. The prepared statement cache can improve the performance of data access.
- IBM i WebSphere Application Server for IBM i Performance Considerations

  This page provides links to information about basic performance considerations for WebSphere Application Server for IBM i.
- IBM i Maximum Performance with WebSphere Application Server V5.1 on iSeries

  Note:

  This book explains how to tune the iSeries server for IBM WebSphere Application Server. The book is divided into several logical tuning tasks, such as tuning the Java virtual machine. You can study each of these logical blocks and apply them independently from other tasks. But, to achieve optimum performance on the system, you should use the systematic approach that is shown in this book.

- IBM i The IBM Virtual Innovation Center™ for Hardware: Education

  Access a variety of Internet-Based Education (IBE) informational modules, each dealing with one specific topic. They are intended to help you incorporate the latest technology into your iSeries applications through multimedia presentations, Q&A areas, examples, and quizzes. You can search for “WebSphere Performance” to find current links on performance education.
- Performance Analysis for Java websites

  Offers clear explanations and expert practical guidance on performance analysis for Java-based websites. It offers extensive appendices, including worksheets for capacity planning, checklists to help you prepare for different stages of performance testing, and a list of performance-test tool vendors.
- WebSphere Application Server Development Best Practices for Performance and Scalability

  Describes development best practices for web applications with servlets, JavaServer Pages files, JDBC connections, and enterprise applications with Enterprise JavaBeans components.

Java™ performance resource

- IBM developerWorks

  Search the IBM developerWorks website for a list of garbage collection documentation, including “Understanding the IBM Java Garbage Collector”, a three-part series. To locate the documentation, search on “sensible garbage collection” in the developerWorks search application.

  Review “Understanding the IBM Java Garbage Collector” for a description of the IBM verbose:gc output and more information about the IBM garbage collector.
- IBM i Basic Java Performance for iSeries servers

  This white paper explains Just-In-Time Compiler coupled with Mixed Mode Interpreter (JIT-MMI), the user class loader verification cache, and memory pool considerations

IBM i

Performance resources

- Tune Java program performance with the IBM Developer Kit for the Java(TM) platform

  This topic describes how you can use the IBM Developer Kit for the Java(TM) platform to optimize the performance of your Java applications.
- Performance management

  This website provides extensive information about the iSeries server performance management.
- Performance Management Resource Library

  This page provides links to information that you can use to optimize server performance. The resource library includes white papers, articles, tools documentation, and more.
- Performance Capabilities Reference Manual

  The Performance Management Library provides links to several editions of the Performance Capabilities Reference Manual. This manual includes information about optimizing performance for DB2 UDB for iSeries servers, web servers and WebSphere products, and Java applications.
- Performance

  This topic provides extensive information about managing and tuning the performance of your iSeries server.
- Applications for performance management

  This topic provides links to several tools that you can use to monitor and manage your iSeries server performance.
- Collection Services

  You can use Collection Services to collect performance data, which you can analyze with other performance tools.
- Performance Tools for your iSeries server

Performance Tools for your iSeries server is a set of tools and commands that you can use to view and analyze performance data in several ways.

- Performance Explorer (PEX)

  Performance Explorer is a data collection tool that helps you identify the causes of performance problems that cannot be identified with the other available tools or general trend analysis. For additional information, also see Performance Explorer (PEX)

- Performance Trace Data Visualizer

  Performance Trace Data Visualizer (PTDV) is a tool for processing, analyzing, and viewing Performance Explorer collection data residing in PEX database files.

- Performance Data Collector tool

  The Performance Data Collector (PDC) tool provides profile information about the programs that run on the iSeries server.

- IBM Performance Management for eServer iSeries

  IBM Performance Management for iSeries (formerly known as PM/400) uses Collection Services to gather the nonproprietary performance and capacity data from your server and then sends the data to IBM for analysis. For additional information, also see IBM Performance Management for eServer iSeries

- iDoctor for IBM i

  iDoctor for IBM i is a suite of applications that can help you monitor performance and troubleshoot common problems on your IBM i server.

Chapter 33. Overview and new features for tuning performance

Use the links provided in this topic to learn about tuning applications and their environment.

There are currently no new tuning performance updates for this release.

Chapter 34. Overview and new features for troubleshooting

Use the links provided in this topic to learn about troubleshooting and problem determination capabilities.

What is new for troubleshooters

This topic provides an overview of new and changed features in troubleshooting tools and support.

Diagnosing problems (using diagnosis tools)

This topic provides a place to start your search for troubleshooting information.

Troubleshooting overview

Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself what happened? A basic troubleshooting strategy at a high level involves:

1. Recording the symptoms.
2. Recreating the problem.
3. Eliminating possible causes.
4. Using diagnostic tools.

Recording the symptoms of the problem

Depending on the type of problem you have, whether it be with your application, your server, or your tools, you might receive a message that indicates something is wrong. Always record the error message that you see. As simple as this sounds, error messages sometimes contain codes that might make more sense as you investigate your problem further. You might also receive multiple error messages that look similar but have subtle differences. By recording the details of each one, you can learn more about where your problem exists.

Recreating the problem

Think back to what steps you were doing that led you to this problem. Try those steps again to see if you can easily recreate this problem. If you have a consistently repeatable test case, you will have an easier time determining what solutions are necessary.

- How did you first notice the problem?
- Did you do anything different that made you notice the problem?
- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If this worked before what has changed? The change can refer to any type of change made to the system, ranging from adding new hardware or software, to configuration changes to existing software.
- What was the first symptom of the problem you witnessed? Were there other symptoms occurring around that point of time?
- Does the same problem occur elsewhere? Is only one machine experiencing the problem or are multiple machines experiencing the same problem?
- What messages are being generated that could indicate what the problem is?

Eliminating possible causes

Narrow the scope of your problem by eliminating components that are not causing the problem. By using a process of elimination, you can simplify your problem and avoid wasting time in areas that are not culprits. Consult the information in this product and other available resources to help you with your elimination process.

See the information on troubleshooting and support to learn more about problem determination tools that are provided by the product.

Using diagnostic tools

As a more advanced task, there are various tools that you can use to analyze and diagnose problems with your system. To learn how to use these tools see the information about using diagnosis tools to diagnose problems. .

What is new for troubleshooters

This version provides many new features for troubleshooting and servicing the product, with a focus on the ability to automatically detect and recover from problems.

- Configuring the memory leak policy

  You can configure WebSphere Application Server to detect, prevent, and take action, if possible, on classloader memory leaks using the memory leak detection policy. For more information about memory leaks, read the Memory Leaks in Java Platform, Enterprise Edition applications topic.
- Cross Component Trace (XCT)

  XCT helps identify the root cause of problems across components, which provides the following benefits:
  - Enables administrators and support teams to follow the flow of a request from end-to-end as it traverses thread or process boundaries, or travels between stack products and WebSphere Application Server.
  - Helps to resolve questions about which component is responsible for a request that fails.
- High Performance Extensible Logging (HPEL)
  - You can use the LogViewer command-line tool to filter records based on the content of log and trace record extensions. The application server automatically creates an appName extension for each log and trace record related to a Java Platform, Enterprise Edition (Java EE) application, indicating the name of that application. The application server also automatically creates a requestID extension for each log and trace record created during the processing of certain types of requests (for example HTTP or JMS requests), indicating the unique ID of that request.
  - Developers can use HPEL to add custom extensions to log and trace records through a log record context API (com.ibm.websphere.logging.hpel.LogRecordContext). When HPEL stores log and trace records, it includes any extensions present in the log record context on the same thread. For example, a developer might write a servlet filter to add important HTTP request parameters to the log record context. While that servlet runs, HPEL adds those extensions to any log and trace records created on the same thread.
- Memory leaks in Java Platform, Enterprise Edition applications

  WebSphere Application Server Version 8.5 provides a top down pattern-based memory leak detection, prevention, and action by watching for suspect patterns in application code at run time. WebSphere Application Server has some means of protection against memory leaks when stopping or redeploying applications. If leak detection, prevention and action are enabled, WebSphere Application Server monitors application and module activity performs diagnostic actions to detect and fix leaks when an application or an individual module stops. This feature helps in increasing application up time with frequent application redeployments without cycling the server.
- Using the IBM Support Assistant Data Collector

  The IBM Support Assistant Data Collector for WebSphere Application Server tool focuses on automatic collection of problem data. It also provides symptom analysis support for the various categories of problems encountered by IBM software products. Information pertinent to a type of problem is collected to help identify the origin of the problem under investigation. The tool assists customers by reducing the amount of time it takes to reproduce a problem with the proper RAS tracing levels set, as well as by reducing the effort required to send the appropriate log information to IBM Support.

Chapter 35. What has changed in this release

These are changes to the default behavior for the application server. Take note of these changes and modify your applications appropriately.

Consult the following topics.

- What has changed for administrators (includes performance)
- What has changed for developers
- IBM i What has changed for installers
- What has changed for security specialists

Note also that you can perform a search for the keyword **trns** to find all topics in which a changed behavior or setting is noted.

Transitioning notes for administration topics

When you upgrade the application server to a newer version, be aware of the following settings that you might want to change.

- APCconfig.jacl script

  The WebSphere Virtual Enterprise command that equates to **APCconfig.jacl** is **xd_APCconfig.jacl**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **xd_APCconfig.jacl** command, which operates the same as the **APCconfig.jacl** command.
- BBSON bulletin board

  The WebSphere Virtual Enterprise command that equates to **manageBBSON.py** is **manageWVEBB.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **manageWVEBB.py** command, which operates the same as the **manageBBSON.py** command.
- dumpIMPState.py script

  The WebSphere Virtual Enterprise command that equates to **dumpIMPState.py** is **dumpXdState.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **dumpXdState.py** command, which operates the same as the **dumpIMPState.py** command.
- Dynamic operations

  In WebSphere Virtual Enterprise, controllers start automatically as highly available managed items unless explicitly suppressed. In Intelligent Management, controllers are dormant until they detect that they are needed. When the controllers are needed, they become active. When the controllers detect that they are no longer needed, they become dormant again. Controllers that are disabled or are in manual mode do not use this detection capability.
- hadmgrAdd command

  The Intelligent Management command that equates to **hadmgrAdd** is **xd_hadmgrAdd**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **xd_hadmgrAdd** command, which operates the same as the **hadmgrAdd** command.
- hadmgrRemove command

  The Intelligent Management command that equates to **hadmgrRemove** is **xd_hadmgrRemove**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **xd_hadmgrRemove** command, which operates the same as the **hadmgrRemove** command.
- manageBBSON.py script

  The WebSphere Virtual Enterprise command that equates to **manageBBSON.py** is **manageWVEBB.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **manageWVEBB.py** command, which operates the same as the **manageBBSON.py** command.
- manageODC.py script

The WebSphere Virtual Enterprise command that equates to **manageODC.py** is **ve_manageODC.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **ve_manageODC.py** command, which operates the same as the **manageODC.py** command.

- PlacementControllerProcs.jacl script

  The WebSphere Virtual Enterprise command that equates to **APCconfig.jacl** is **xd_APCconfig.jacl**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **xd_APCconfig.jacl** command, which operates the same as the **APCconfig.jacl** command.

- useBBSON.py script

  The Intelligent Management command that equates to **useBBSON.py** is **useWVEBB.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **useWVEBB.py** command, which operates the same as the **useBBSON.py** command.

- workclassoperations.py script

  The WebSphere Virtual Enterprise command that equates to **IMPPYModules.py** is **XDPYModules.py**. If you are making the transition from WebSphere Virtual Enterprise, you can continue to use the **XDPYModules.py** command, which operates the same as the **IMPPYModules.py** command.

Transitioning notes for development topics

When you upgrade the application server to a newer version, be aware of the following settings that you might want to change.

- Configuring JavaServer Faces implementation

  The Application Server JSF engine determines if the SUN Reference Implementation (RI) 1.2 or Apache MyFaces 2.0 is used from the Application Server run time. If either is used, the correct listener class is registered with the web container. You do not need to add the com.sun.faces.ConfigureListener or the org.apache.myfaces.StartupConfigureListener to your `web.xml` file.

- Configuring the SCA JMS binding

  Differences between the OSOA and OASIS JMS binding specifications might affect migration of applications from OSOA to OASIS. Some commonly encountered differences include:

  – The validation schemas used for the OASIS applications are defined by the OASIS SCA specification. The OASIS binding schema definition mandates that binding elements appear in the exact order as defined in the schema. For OSOA applications, element order does not affect validation so, when migrating applications from OSOA, ensure the binding elements are in the order specified by the OASIS binding schema.

  – For the `<destination>`, `<connectionFactory>`, and `<activationSpec>` elements, the name attribute is `jndiName` in OASIS.

  – In OASIS composites, callback references must specify a destination element. The specified destination is used only if the destination cannot be determined from the scaCallbackDestination or JMSReplyTo properties of the service request message. In OSOA composites, callback references can omit the destination element if it is known that one of the properties is always set.

  For information on other differences, consult the OASIS specification for additional information about specific elements.

- Migration of JPA applications and bean validation

  The Bean Validation API and implementation are included in the Version 8.x product. If you want to deploy the feature pack applications on Version 8.x, be aware of the following changes in configuration and runtime behaviors:

  – In Version 7.0, the system property, com.ibm.websphere.validation.api.jar.path, is used to direct the feature pack run time to locate the Bean Validation API JAR file. In Version 8.x, the Bean Validation API JAR file is built into the product installation, therefore, this system property is ignored. However, the bean validation provider can still be overridden in Version 7 and Version 8.x when packaged in application or shared libraries. The third bullet describes this usage scenario.

- The bean validation implementation in WebSphere Application Server is automatically the effective default bean validation provider.
- If standard bean validation features are used by the JPA application, the com.ibm.websphere.validation.api.jar.path system property, and user-supplied bean validation provider can be removed from application deployment.
- If non-specification compliant or provider-specific features are used by an application, the bean validation provider that supports the required features must be packaged, configured, and deployed as a typical, third-party service provider in the user application, the application shared library, and the server associated shared library. Read more about shared library usage in applications for more information.

- Migration scenario for the getHeaderNames method

  To ensure the most successful migration, the return type of the StoredResponse getHeaderNames method was changed to com.ibm.websphere.servlet.response.CollectionEnumerationHybrid<String>. This type implements both Collection<String> and Enumeration.
- "Runtime considerations for SIP application developers" on page 644

  When a SIP application sends a request to a SIP URI over Transport Layer Security (TLS) in version 6.1, the request URI scheme changes from "sip" to "sips." In the current version, the scheme does not change. You can reverse the new behavior by changing the application code. With a "sips" URI, the behavior remains the same after upgrading from version 6.1 to 7.0 or above. See the information center topic Premigration considerations for more information.

Transitioning notes for deployment topics

When you upgrade the application server to a newer version, be aware of the following settings that you might want to change.

- BLAManagement command group for the AdminTask object using wsadmin scripting

  In the WebSphere Application Server Version 7 Feature Pack for OSGi Applications and Java Persistence API 2.0, bundle changes to the asset are applied by restarting the business-level application. In Version 8.x, these changes are applied by updating the composition unit. The new approach in Version 8.x means that many bundle changes can be applied in place, without restarting the running business-level application. To enable this new approach, the **UpdateAppContentVersionsStep** parameter has been replaced with the **UpdateAppContentVersions** parameter, and instead of restarting the business-level application you run the **editCompUnit** command with the **CompUnitStatusStep** parameter.
- SCA programming model support in OSGi applications

  If your application previously used SDO version 1.0.0, change the Import-Package to:

  ```
  Import-Package: commonj.sdo;version="[2.0.0,3.0.0)"
  ```

Transitioning notes for security topics

When you upgrade the application server to a newer version, be aware of the following settings that you might want to change.

- SSL cipher specifications

  To improve security, IBM HTTP Server Version 8.0 disables weak SSL ciphers, export SSL ciphers, and the SSL version 2 protocol by default. SSL Version 2, weak ciphers, and export ciphers are generally unsuitable for production SSL workloads on the internet and are flagged by security scanners. To enable ciphers, use the SSLCipherSpec directive.

Chapter 36. WebSphere Application Server roles and goals

There are several different computing roles that members of your organization might undertake when working with WebSphere Application Server.

Enterprise architect

The enterprise architect provides overall leadership for all architectural and technological matters with respect to the company's IT environment.

Solution architect

The solution architect designs and coordinates a new solution, application or component with end-to-end responsibility including both hardware and software elements.

The main goal of the solution architect is to design a solution that supports the specification set by the enterprise architect.

System administrator

The system administrator is responsible for managing systems and software, and for installing operating system upgrades and middleware products in many accounts.

The system administrator installs and configures appropriate hardware and software (including middleware) to implement the design provided by the solution architect. Additionally the system administrator monitors and maintains the configured system, modifying and removing previously configured objects as and when required.

Application developer

The application developer creates business applications.

The goal of the application developer is to develop applications that provide the business services described by the solution architect.

Chapter 37. Fast paths for WebSphere Application Server

Use the paths in this topic to deploy applications quickly and easily. This topic provides links that pinpoint the relevant information for reaching your goals quickly. It also describes the audience roles and tasks assumed by this documentation. The fast paths in this topic are intended to help you gain a little experience. The fast paths do not showcase the advanced product features that some users need or want to use in their production environments.

About this task

Deploying any type of application involves the following tasks.

Procedure

1. Install the product.

   The simplest scenario is to perform a typical installation of a single application server.

   IBM i You can use the WebSphere Application Server, Express for IBM i product. It has a single application server and is available at no cost with IBM i Version 5 Release 4 and later.
2. Obtain or develop your application.

   In the simplest scenario, you already have a packaged web application that is compliant with Java Platform, Enterprise Edition (Java EE), perhaps from a vendor with whom you work. Other than that, a web application is the most simple type of application to develop. See topics on developing web applications, or refer to product samples.
3. Deploy and test your application.

   In the simplest scenario, you will use the application installation wizard available in the administrative console. See topics on installing applications for a detailed walkthrough of this task. The task describes many contingencies, but many of these steps can be disregarded if:
   - You are deploying a web application.
   - You accept the default settings whenever possible.
   - Your application does not require data access.
   - Your application does not require security.

   Also, in many situations, you do not need to modify the default application server configuration.
4. Administer your deployed application.

   In the simplest scenario, you monitor your application with the Tivoli Performance Viewer functionality built into the administrative console. For an overview of this task, see topics on Monitoring performance.

Results

Now you should have some insight into the task flow for successfully deploying your applications.

To solidify and expand your understanding, view the following table. It shows the user role and task assumptions by which this documentation is organized, for predictability. If you know what role or task you are performing, you can disregard (at least temporarily) documentation that is labeled for other roles or tasks. By their nature, role and task models are simplified compared to reality. Also, one person might perform many roles and tasks in the course of a day.

Table 106. Mapping of user roles to user tasks. Each user role performs at least one user task.

User role	User tasks
Installer	Install application serving environment Administer applications and environment Migrate deployed applications and their environment
Administrator	Administer applications and environment Deploy applications into production Monitor and tune applications and their environment
Developer	Develop or migrate application code Assemble applications for deployment Deploy applications for testing
Security expert	Secure applications and their environment
Troubleshooter	Use tools to troubleshoot problems

Chapter 38. Release notes

Links are provided to a description of the new functionality, the product support website, the product documentation, and to last-minute updates, limitations, and known problems for IBM WebSphere Application Server Version 8.5 products.

- “Accessing last-minute updates, limitations, and known problems”
- “Accessing hardware and software requirements”
- “Accessing product documentation”
- “Accessing the product support website”
- “Contacting IBM Software Support”

Accessing last-minute updates, limitations, and known problems

Release notes are available at the following URL:

- IBM i WebSphere Application Server Version 8.5

Accessing hardware and software requirements

- The hardware and software requirements for the WebSphere Application Server products are provided on the Detailed system requirements web page.

Accessing product documentation

The following documentation is installed with this product.

Help files

The help files provide detailed instructions for completing tasks and for specifying settings in the graphical systems management tools. Use the help menus, links, or buttons provided in the tool interfaces to access the help files.

Documentation on the web

For the entire documentation set for all WebSphere Application Server products, including the information center and Adobe Acrobat PDF versions of the information, go to the WebSphere Application Server library web page. All product documentation is in the information center, including versions of the installed help files.

Accessing the product support website

- To search for the latest troubleshooting tips, downloads, fixes, and other support-related information, go to the WebSphere Application Server support page.

Contacting IBM Software Support

If you encounter a problem with this product, first try the following actions:

- Following the steps described in the product documentation
- Looking for related documentation in the online help
- Looking up error messages in the message reference

If you cannot resolve your problem by any of the preceding methods:

1. Go to the WebSphere Application Server support page.
2. Expand the Contact Support section.
   - Click "Exchanging information with IBM Tech Support", and follow the information collection instructions provided.

- Click "Directory of worldwide contacts" to find the appropriate contact information for your geography.

Chapter 39. WebSphere platform and related software

This article provides information on other WebSphere and IBM products.

What is WebSphere?

IBM WebSphere® is the leading software platform for e-business on demand™. Successful industry leaders choose this adaptable, open platform to succeed in quickly and reliably delivering business results today knowing it gives them the freedom to grow their e-business in the future. Providing a full range of middleware software and support offerings, the WebSphere platform is infrastructure software that enables companies to develop, deploy, and integrate next-generation e-business applications from simple web publishing through enterprise-scale transaction processing.

WebSphere software transforms the way businesses manage customer, partner, and employee relationships. For example, you can use it to create compelling Web experiences, extend applications to incorporate mobile devices, and build electronic e-marketplaces. WebSphere software is all about the three fundamental aspects of e-business on demand:

- **Foundation & Tools.** Rely upon a high quality foundation to rapidly build and deploy applications for high-performance e-business on demand.
- **Business Portals.** Enhance customer, partner, employee, and supplier user experiences for optimal satisfaction.
- **Business Integration.** Integrate applications and automate business processes for operational efficiency and business flexibility.

WebSphere software platform

This page is the main web page for finding information about products in the WebSphere brand.

WebSphere Extended Deployment

WebSphere Extended Deployment offers a dynamic, goals-directed, high-performance environment for running mixed application types and workload patterns in WebSphere. With these capabilities, you can optimize the resource utilization and management of your IT infrastructure, while enhancing the quality of service for your business-critical applications.

Chapter 40. Guided activities for the administrative console

The topic describes the *guided activities* that are available in the administrative console. Guided activities lead you through common administrative tasks that require you to visit multiple administrative console pages.

Table 107. Quick reference: Accessing the guided activities. The following table gives you the web address for the guided activities in the administrative console.

The guided activities are available from the main page of the administrative console. The page is displayed after you log into the administrative console. To open the console, enter this web address in your web browser: `http://`*`your_fully_qualified_server_name`*`:9060/ibm/console` Depending on your configuration, your web address might differ. Other factors can affect your ability to access the console. See Starting and logging off the administrative console for details, as needed.

Guided activities display each administrative console page that you need to perform a task, surrounded by the following information to help you perform the task successfully.

- An introduction to the task, introducing essential concepts and describing when and why to perform the task
- Other tasks to do before and after performing the task
- The main steps to complete during this task
- Hints and tips to help you avoid and recover from problems
- Links to field descriptions and extended task information in the online documentation

Chapter 41. Tutorials

This topic describes how to find tutorials and their accompanying samples, for learning how to accomplish your goals with the product.

IBM Education Assistant tutorials
The IBM Education Assistant site provides education resources that you can use at your convenience.

developerWorks tutorials and training
The Tutorials and Training page of developerWorks provides tutorials and other training resources that you can use at your convenience.

Chapter 42. Accessing the samples

The product offers samples that demonstrate common enterprise application tasks. Many samples also provide instructions for deployment and coding examples.

The product provides samples in two ways:

Plants By WebSphere sample installed with the product

If you select to install samples when installing the product and when creating an application server profile, the Plants By WebSphere application is included with the product. The application demonstrates several Java Platform, Enterprise Edition (Java EE) functions, using an online store that specializes in plant and garden tool sales.

See Installing the Plants By WebSphere sample.

Samples downloadable from the Samples, Version 8.5 information center

The product provides component-specific samples that you can download at any time from a download site.

- Available samples
- Downloading samples

Installing the Plants By WebSphere sample

To install the Plants By WebSphere sample, perform the following steps.

1. Install the product.

   Plants By WebSphere sample is installed in the *app_server_root*/`samples` directory. A Plants By WebSphere pre-built enterprise archive named `pbw-ear.ear` is in the `/samples/PlantsByWebSphere/pbw-ear/target` directory.

   Installation instructions are in the `/samples/PlantsByWebSphere/docs` directory.

   You can build or modify the sample source code to support your project. The source code is in a `src` directory.
2. Start the application server.

Available samples

Samples that you can download include, for example, the following materials:

Communications Enabled Applications (CEA) samples

The CEA sample applications provide two main services, telephony access and multi-modal web interaction. Use this collection of sample applications to explore the services and to use as a starting point when developing your own communication enabled applications.

OSGi samples

The OSGi samples help you develop and deploy modular applications that use both Java EE and OSGi technologies.

XML samples

The XML samples demonstrate use of the XML API and supported specifications.

Internationalization service sample

The Internationalization service sample demonstrates how to use the internationalization service in Java EE applications, specifically within servlets and enterprise beans.

Web services samples

These samples demonstrate both Java API for XML-based RPC (JAX-RPC) and Java API for XML Web Services (JAX-WS) web services that use Java Platform, Enterprise Edition (Java EE) beans and JavaBeans components.

The JAX-WS web service samples demonstrate the implementation of one-way and two-way web services that highlight the use of web services standards such as WS-Addressing (WS-A) , WS-Reliable Messaging (WS-RM), and WS-Secure Conversation (WS-SC) and the SOAP Message Transmission Optimization Mechanism (MTOM) technology.

Service Data Objects (SDO) sample

This sample demonstrates data access to a relational database through Service Data Objects (SDO) and Java DataBase Connectivity (JDBC) Mediator technologies.

Downloading samples

You can download samples from the **Samples, Version 8.5** information center.

1. Go to the **Samples, Version 8.5** information center.
2. Determine which samples you want to download.
3. On the **Downloads** tab for the samples that you want, click a **Download Sample** link.
4. In the authentication window, click **OK**.
5. Download the compressed file, or individual sample files, to a directory on your workstation.

   You might create the `/samples/`*`sample_type`* directory path on your workstation and download the sample files to that directory path.

Many sample compressed files have an `/installableApps` directory that contains deployable prebuilt archives. Other directories contain files such as sample-specific source archives, scripts, and instructions for building deployable archives.

Limitations of the samples

- The samples are for demonstration purposes only.

  The code that is provided is not intended to run in a secured production environment. The samples support Java 2 Security, therefore the samples implement policy-based access control that checks for permissions on protected system resources, such as file I/O.

  The samples also support administrative security.

Additional samples and examples

Samples on developerWorks

Additional product samples are available on WebSphere developerWorks

Samples in tutorials

Many product tutorials rely on sample code. To find tutorials that demonstrate specific technologies, browse the links in “Tutorials” on page 12.

Examples in the product documentation

The product documentation contains many code snippets and examples. To locate these examples easily, see the developer examples in the **Reference** section of the information center navigation for the product edition that you are using.

IBM i **IBM Telephone Directory**

The IBM Telephone Directory business application is shipped separately from the product. For information about obtaining and using the IBM Telephone Directory application, see IBM Telephone Directory V5.2 in the topic on e-business and web serving in the IBM i information center.

Chapter 43. Using the administrative clients

The product provides a variety of administrative clients for deploying and administering your applications and application serving environment, including configurations and logical administrative domains.

Procedure

- Using the administrative console

  The administrative console is a graphical, browser-based tool.
- Getting started with wsadmin scripting

  Scripting is a non-graphical alternative that you can use to configure and administer your applications and application serving environment. The WebSphere Application Server **wsadmin** tool provides the ability to run scripts. The wsadmin tool supports a full range of product administrative activities.

  IBM i See also Configuring Qshell to run WebSphere scripts using wsadmin scripting.
- Using Ant to automate tasks

  To support using Apache Ant with Java Platform, Enterprise Edition (Java EE) applications running on IBM WebSphere Application Server, the product provides a copy of the Ant tool and a set of Ant tasks that extend the capabilities of Ant to include product-specific functions.
- Using administrative programs (JMX)

  The product supports access to the administrative functions through a set of Java classes and methods, under the Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the other administrative clients. You also can extend the basic product administrative system to include your own managed resources.
- Using command-line tools

  Several command-line tools are available that you can use to start, stop, and monitor WebSphere server processes and nodes. These tools work on local servers and nodes only. They cannot operate on a remote server or node.

  IBM i See also Configuring Qshell to run WebSphere scripts using wsadmin scripting.

Chapter 44. Specifications and API documentation

The WebSphere Application Server product supports various industry standards. This topic lists the specifications and application programming interface (API) documentation supported in current and previous product releases.

Components

- Any application type
- Web applications
- Portlet applications
- SIP applications
- EJB applications
- OSGi applications
- Client applications
- Web services
- Service integration
- Data access resources
- Messaging resources
- Mail, URLs, and other Java EE resources
- Security
- Web Services Security
- Naming and directory
- Object Request Broker
- Transactions
- WebSphere extensions
- Administration

The **Version 8.5** column in the tables lists the latest specification level that the product supports. However, support for specifications is compatible with earlier versions of the product; the Version 8.5 product supports all specifications that are listed for Version 6.0 through Version 8.5. For example, for any application type, the Version 8.0 product supports Java EE 5 and 6 and J2EE 1.2, 1.3, and 1.4. The word "New" beside a specification indicates that the product first supported the specification in that product version.

Any application type

Table 108. Supported specifications for any application type. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Platform, Enterprise Edition (Java EE) specification Prior to Java EE 5, the specification name was Java 2 Platform, Enterprise Edition (J2EE).	Java EE 6 (JSR 316)	Java EE 6 (JSR 316) New	Java EE 5 New	J2EE 1.4	J2EE 1.4 New J2EE 1.3 J2EE 1.2
Java Platform, Standard Edition (Java SE) specification Prior to Java SE 6, the specification name was Java 2 Platform, Standard Edition (J2SE).	Java SE 7 New	Java SE 6	Java SE 6 New	J2SE 5	J2SE 1.4.2
ISO 8859 specifications	ISO 8859 applies to these versions.				

Web applications

Table 109. Supported specifications for web applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Servlet specification (JSR 154, JSR 53 and JSR 315)	Java Servlet 3.0	Java Servlet 3.0 New	Java Servlet 2.5 New	Java Servlet 2.4	Java Servlet 2.4 New Java Servlet 2.3
JavaServer Faces (JSF) specification (JSR 252 and 127)	Apache MyFaces - JSF 2.0	Apache MyFaces - JSF 2.0 New	Sun Reference Implementation - JSF 1.2 Apache MyFaces 1.2 - JSF 1.2	JSF 1.1	JSF 1.0
JavaServer Pages (JSP) specification (JSR 245, JSR 152, and JSR 53)	JSP 2.2	JSP 2.2 New	JSP 2.1 New	JSP 2.0	JSP 2.0 New JSP 1.2

Portlet applications

Table 110. Supported specifications for portlet applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Portlet specification	Portlet 2.0 (JSR 286)	Portlet 2.0 (JSR 286)	Portlet 2.0 (JSR 286) New	Portlet 1.0 (JSR 168)	Not applicable. The product supports portlets in Version 6.1.

Session Initialization Protocol applications

Table 111. Supported specifications and APIs for SIP applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Session Initiation Protocol (SIP) Servlet API For a complete list of SIP and SIP proxy standards, see “SIP industry standards compliance” on page 642.	SIP 1.1 (JSR 289)	SIP 1.1 (JSR 289) New	SIP 1.1 (JSR 289) New for Feature Pack for CEA 1.0	SIP 1.0 (JSR 116)	Not applicable. The product supports SIP in Version 6.1.

Enterprise bean (EJB) applications

Table 112. Supported specifications and APIs for EJB applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Enterprise JavaBeans (EJB) specification	EJB 3.1	EJB 3.1 New	EJB 3.0	EJB 3.0 New for Feature Pack for EJB 3.0	EJB 2.1 New EJB 2.0 EJB 1.1
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New JDBC 2.1 and Optional Package API (2.0)
Java Message Service (JMS) specification	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1 New
Java Persistence API (JPA) specification	JPA 2.0	JPA 2.0	JPA 2.0 New for Feature Pack for OSGi and JPA 2.0	JPA 1.0 New for Feature Pack for EJB 3.0	Not applicable

OSGi applications

Table 113. Supported specifications and APIs for OSGi applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
OSGi Service Platform specification	OSGi Service Platform Release 4 Version 4.2	OSGi Service Platform Release 4 Version 4.2	OSGi Service Platform Release 4 Version 4.2 New for Feature Pack for OSGi and JPA 2.0	Not applicable	Not applicable
OSGi Alliance RFC-0112 Bundle Repository specification	OSGi Alliance RFC-0112 (Draft)	OSGi Alliance RFC-0112 (Draft)	OSGi Alliance RFC-0112 (Draft) New for Feature Pack for OSGi and JPA 2.0	Not applicable	Not applicable

Client applications

Table 114. Supported specifications and APIs for client applications. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Web Start architecture	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2	Java Web Start 1.4.2 New

Web services

Table 115. Supported specifications and APIs for web services. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Architecture for XML Binding (JAXB) specification	JAXB 2.2	JAXB 2.2 New	JAXB 2.1 New	JAXB 2.0 New for Feature Pack for Web Services	Not applicable
Java Architecture for XML Binding (JAXB) Reference Implementation Vendor Extensions Runtime Properties specification	JAXB 2.2 RI Vendor Extensions	JAXB 2.2 RI Vendor Extensions New	JAXB 2.1 RI Vendor Extensions New	JAXB 2.0 RI Vendor Extensions New for Feature Pack for Web Services	Not applicable
Java API for XML Processing (JAXP) specification	1.4 Included in Java SE 6.	1.4 Included in Java SE 6.	1.4 Included in Java SE 6.	1.3 Included in J2SE 5.	1.2 Maintenance release of JSR 63
Java API for XML Registries (JAXR) specification	JAXR 1.0	JAXR 1.0	JAXR 1.0	JAXR 1.0	JAXR 1.0 New
Java API for XML-based RPC (JAX-RPC) specification	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1	JAX-RPC 1.1 New
Java API for RESTful Web Services (JAX-RS) specification	JAX-RS 1.1	JAX-RS 1.1 New			
Java API for XML Web Services (JAX-WS) specification	JAX-WS 2.2	JAX-WS 2.2 New	JAX-WS 2.1 New	JAX-WS 2.0 New for Feature Pack for Web Services	Not applicable
SOAP	SOAP 1.2	SOAP 1.2	SOAP 1.2	SOAP 1.2 New for Feature Pack for Web Services	SOAP 1.1
SOAP with Attachments API for Java (SAAJ) Specification	SAAJ 1.3	SAAJ 1.3	SAAJ 1.3	SAAJ 1.3 New for Feature Pack for Web Services	SAAJ 1.2 New
SOAP over Java Message Service (SOAP over JMS)	W3C SOAP over JMS 1.0	W3C SOAP over JMS 1.0	W3C SOAP over JMS 1.0 (submission draft)		
SOAP Message Transmission Optimization Mechanism (MTOM)	MTOM 1.0	MTOM 1.0	MTOM 1.0	MTOM 1.0 New for Feature Pack for Web Services	Not applicable

Table 115. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Streaming API for XML (StAX)	StAX 1.0	StAX 1.0	StAX 1.0	StAX 1.0 New for Feature Pack for Web Services	Not applicable
Universal Description, Discovery and Integration (UDDI)	UDDI 3.0	UDDI 3.0	UDDI 3.0	UDDI 3.0	UDDI 3.0 New
W3C XML Schema	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2	• XML Schema 1.0 • XML Schema Part 1 • XML Schema Part 2
Web Services Addressing (WS-Addressing) For more information, see "Web Services Addressing version interoperability" on page 787.	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • 1.0 Core • 1.0 SOAP Binding • 1.0 Metadata • WS-Addressing WSDL Binding, W3C Candidate Recommendation • WS-Addressing, W3C Submission	WS-Addressing 1.0 family of specifications: • Core • SOAP Binding • WSDL Binding • WS-Addressing WSDL Binding, W3C Last Call • WS-Addressing, W3C Submission	Not applicable
Web Services Atomic Transaction (WS-AT)	WS-AT 1.2	WS-AT 1.2	WS-AT 1.1 New WS-AT 1.2 New	WS-AT 1.0	WS-AT 1.0 New
Web Services Business Activity (WS-BA)	WS-BA 1.2	WS-BA 1.2	WS-BA 1.1 New WS-BA 1.2 New	WS-BA 1.0	Not applicable
Web Services Coordination (WS-COOR)	WS-COOR 1.2	WS-COOR 1.2	WS-COOR 1.1 New WS-COOR 1.2 New	WS-COOR 1.0	WS-COOR 1.0 New
Web Services Description Language (WSDL)	WSDL 1.1	WSDL 1.1	WSDL 1.1	WSDL 1.1	WSDL 1.1

Table 115. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Web Services for Java Platform, Enterprise Edition (Java EE) (JSR 109) Prior to Web Services for Java EE, the specification name was Web Services for Java 2 Platform, Enterprise Edition (J2EE).	JSR 109 1.3	JSR 109 1.3 New	JSR 109 1.2 New	JSR 109 1.1	JSR 109 1.1 New
Web Services Interoperability Organization (WS-I) Basic Profile	WS-I Basic Profile 1.2 WS-I Basic Profile 2.0	WS-I Basic Profile 1.2 WS-I Basic Profile 2.0	WS-I Basic Profile 1.2 (draft) WS-I Basic Profile 2.0 (draft)	WS-I Basic Profile 1.2 (draft) New for Feature Pack for Web Services WS-I Basic Profile 2.0 (draft) New for Feature Pack for Web Services	WS-I Basic Profile 1.1 New
Web Services-Interoperability (WS-I) Attachments Profile	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0	WS-I Attachments 1.0 New
Web Services Interoperability (WS-I) Reliable Secure Profile (RSP) Prior to WS-I RSP, the specification was named Reliable Asynchronous Messaging Profile (RAMP)	WS-I RSP 1.0	WS-I RSP 1.0	RAMP 1.0	RAMP 1.0 New for Feature Pack for Web Services	Not applicable
Web Services Invocation Framework (WSIF)	WSIF	WSIF	WSIF	WSIF	WSIF
Web Services Metadata for the Java Platform (JSR 181)	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform	Web Services Metadata for the Java Platform New for Feature Pack for Web Services	Not applicable
Web Services Notification (WS-Notification)	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	WS-Notification 1.3 family of specifications: • WS-BaseNotification 1.3 • WS-BrokeredNotification 1.3 • WS-Topics 1.3	Not applicable

Table 115. Supported specifications and APIs for web services (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Web Services Policy (WS-Policy) specification	Web Services Policy 1.5 Web Services Addressing 1.0 - Metadata Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 WS-SecurityPolicy 1.2	Web Services Policy 1.5 Web Services Addressing 1.0 - Metadata Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 WS-SecurityPolicy 1.2	Web Services Policy 1.5 New Web Services Addressing 1.0 - Metadata New Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1 New Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1 New WS-SecurityPolicy 1.2 New	Not applicable	
Web Services Reliable Messaging	WS-MakeConnection Version 1.0	WS-MakeConnection Version 1.0	WS-MakeConnection Version 1.0 New	WS-ReliableMessaging 1.0 and WS-ReliableMessaging 1.1. New for Feature Pack for Web Services	Not applicable
Web Services Resource Framework (WSRF)	WSRF 1.2	WSRF 1.2	WSRF 1.2	WSRF 1.2 New	Not applicable
XML-binary Optimized Packaging (XOP)	XOP 1.0	XOP 1.0	XOP 1.0	XOP 1.0 New for Feature Pack for Web Services	Not applicable

Service integration

Table 116. Supported specifications and APIs for service integration. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New

Data access resources

Table 117. Supported specifications and APIs for data access resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java DataBase Connectivity (JDBC) API	JDBC 4.1 New	JDBC 4.0	JDBC 4.0 New	JDBC 3.0	JDBC 3.0 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0
Service Data Objects (SDO) specification				SDO 1.0	SDO 1.0 New

Messaging resources

Table 118. Supported specifications and APIs for messaging resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Message Service (JMS)	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1	JMS 1.1 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0

Mail, URLs, and other Java EE resources

Table 119. Supported specifications and APIs for mail, URLs, and other Java EE resources. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
JavaMail API documentation (JSR 919)	JavaMail 1.4	JavaMail 1.4	JavaMail 1.4 New	JavaMail 1.3	JavaMail 1.3 New
URL API documentation	URL 1.4.2	URL 1.4.2	URL 1.4.2	URL 1.4.2	URL 1.4.2 New
JavaBeans Activation Framework (JAF) Specification	JAF 1.1	JAF 1.1	JAF 1.1 New	JAF 1.0.2	JAF 1.0.2 New
W3C Architecture - Naming and Addressing: URIs, URLs	W3C Naming and Addressing applies to these versions.				

Security

Table 120. Supported specifications and APIs for security. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java 2 Security Manager	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.5	Java 2 Security Manager 1.4 New

Table 120. Supported specifications and APIs for security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Authentication and Authorization Service (JAAS)	JAAS 2.0 applies to these versions.				
Java Authorization Contract for Containers (JACC)	JACC 1.4	JACC 1.4	JACC 1.4 New	JACC 1.0	JACC 1.0 New
Java Authentication Service Provider Interface for Containers (JASPI)	JASPI 1.0	JASPI 1.0	Not applicable	Not applicable	Not applicable
Common Secure Interoperability Version 2 (CSIv2) specification This is an Object Management Group (OMG) CORBA/IIOP specification.	CSI 2.0 applies to these versions.				
Secure Sockets Layer (SSL) configuration The product uses Java Secure Sockets Extension (JSSE) as the SSL implementation for secure connections. JSSE is part of the Java 2 Standard Edition (J2SE) specification and is included in the IBM implementation of the Java Runtime Extension (JRE) specification.	JSSE 5.0	JSSE 5.0	JSSE 5.0	JSSE 5.0 New	JSSE 1.0.3
Java Generic Security Service (JGSS) Use JGSS with the Kerberos Network Authentication Service, Version 5	JGSS 1.0.1 applies to these versions.				
The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)	SPNEGO 1.0 applies to these versions.				
Java Cryptographic Extension (JCE) specification	JCE 1.0 applies to these versions.				
Java Certification Path (CertPath) API	CertPath 1.1	CertPath 1.1	CertPath 1.1	CertPath 1.1 New	CertPath 1.0

Web Services Security

Table 121. Supported specifications and APIs for Web Services Security. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Canonical XML	Canonical XML 1.0 applies to these versions.				
Decryption Transform for XML Signature	Decryption Transformation for XML Signature applies to these versions. .				
Exclusive XML Canonicalization	Exclusive XML Canonicalization 1.0 applies to these versions.				
OASIS Web Services Security: SOAP Message Security (WS-Security)	WS-Security 1.1	WS-Security 1.1	WS-Security 1.1	WS-Security 1.1 New for Feature Pack for Web Services	WS-Security 1.0

Table 121. Supported specifications and APIs for Web Services Security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
OASIS Web Services Security: Kerberos Token Profile	Kerberos Token Profile 1.1	Kerberos Token Profile 1.1	Kerberos Token Profile 1.1 New	Not applicable	
OASIS Web Services Security: SAML Token Profile 1.1 **Note:** WebSphere Application Server supports this specification in reference to the SAML Version 1.1 and 2.0 assertions within SOAP messages only.	SAML Version 1.1 and 2.0 assertions	SAML Version 1.1 and 2.0 assertions	SAML Version 1.1 and 2.0 assertions		
OASIS Web Services Security: Username Token Profile	Username Token Profile 1.1	Username Token Profile 1.1	Username Token Profile 1.1	Username Token Profile 1.1 New for Feature Pack for Web Services	Username Token Profile 1.0 New
OASIS Web Services Security: X.509 Token Profile	X.509 Token Profile 1.1	X.509 Token Profile 1.1	X.509 Token Profile 1.1	X.509 Token Profile 1.1 New for Feature Pack for Web Services	X.509 Token Profile 1.0 New
Web Services Interoperability Organization (WS-I) Basic Security Profile	WS-I Basic Security Profile 1.1	WS-I Basic Security Profile 1.1	WS-I Basic Security Profile 1.1 New	WS-I Basic Security Profile 1.0	Not applicable
Web Services Interoperability Organization (WS-I) Reliable Secure Profile	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft)	WS-I Reliable Secure Profile 1.0 (draft) New for Feature Pack for Web Services	Not applicable
Web Services Secure Conversation (WS-SecureConversation)	OASIS WS-SecureConversation 1.3	OASIS WS-SecureConversation 1.3	OASIS WS-SecureConversation 1.3 New	OASIS WS-SecureConversation 1.0 (draft submission) New for Feature Pack for Web Services	Not applicable

Table 121. Supported specifications and APIs for Web Services Security (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Web Services Trust	OASIS WS-Trust 1.3	OASIS WS-Trust 1.3	OASIS WS-Trust 1.3 New	OASIS WS-Trust 1.1 (draft) New for Feature Pack for Web Services	Not applicable
XML Signature Syntax and Processing	XML Signature Syntax and Processing applies to these versions.				
XML Encryption Syntax and Processing	XML Encryption Syntax and Processing applies to these versions.				

Naming and directory

Table 122. Supported specifications and APIs for naming and directory. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java Naming and Directory Interface (JNDI) Specification See also "JNDI support in WebSphere Application Server" on page 467.	JNDI on Java SE 7 New	JNDI on Java SE 6	JNDI on Java SE 6 New	JNDI on J2SE applies to these versions.	
Common Object Request Broker: Architecture and Specification (CORBA) specification This is an Object Management Group (OMG) Interoperable Naming (CosNaming) specification.	CORBA 2.4 applies to these versions.				
Interoperable Naming Service specification This is an OMG CosNaming specification.	Interoperable Naming Service				
Naming Service specification This is an OMG CosNaming specification.	Naming Service applies to these versions.				

Object Request Broker

The Object Request Broker (ORB) component follows the Common Object Request Broker Architecture (CORBA) specifications supported by Java 2 Platform, Standard Edition (J2SE). The Object Management Group (OMG) produces the specifications.

Versions 6.1 and later use the J2SE 5.0 specifications that are listed in *Official Specifications for CORBA support in J2SE 5.0* at http://download.oracle.com/javase/1.5.0/docs/guide/idl/compliance.html.

Version 6.0.x uses the J2SE 1.4 specifications that are listed in *Official Specifications for CORBA support in J2SE 1.4* at http://download.oracle.com/javase/1.4.2/docs/api/org/omg/CORBA/doc-files/compliance.html.

Table 123. Supported specifications and APIs for ORB. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Common Object Request Broker Architecture (CORBA) specifications	CORBA 2.3.1 applies to these versions.				
Revised IDL to Java language mapping	Revised IDL to Java language mapping applies to these versions.				
New IDL to Java Mapping Chapter	New IDL to Java Mapping Chapter applies to these versions.				
Updated Java to IDL Mapping specification	Updated Java to IDL Mapping applies to these versions.				
Interoperable Naming Service revised chapters	Interoperable Naming Service revised chapters applies to these versions.				
Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification	Object Reference Template Final Adopted specification New	Not applicable
Portable Interceptors specification	Not applicable	Not applicable	Not applicable	Not applicable	Portable Interceptors specification

Transactions

Table 124. Supported specifications and APIs for transactions. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
CORBA Object Transaction Service (OTS) specification	OTS 1.4	OTS 1.4	OTS 1.4	OTS 1.4	OTS 1.4 New
Java EE Connector Architecture (JCA) resource adapter	JCA 1.6 (JSR 322)	JCA 1.6 (JSR 322) New	JCA 1.5	JCA 1.5	JCA 1.5 New JCA 1.0
Java Transaction API (JTA) specification	JTA 1.1	JTA 1.1	JTA 1.1 New	JTA 1.0.1B	JTA 1.0.1B New
Java Transaction Service (JTS) specification	JTS 1.0 applies to these versions.				

WebSphere extensions

Table 125. Supported specifications and APIs for WebSphere extensions. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
ActivitySession service and Last Participant Support					
J2EE Activity Service for Extended Transactions (JSR 95)	JSR 95 applies to these versions.				
Java Transaction API (JTA) specification	JTA 1.1	JTA 1.1	JTA 1.1 New	JTA 1.0.1B New	JTA 1.0.1
Internationalization (i18n)					

Table 125. Supported specifications and APIs for WebSphere extensions (continued). The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
J2SE internationalization documentation	J2SE Internationalization 5.0	J2SE Internationalization 5.0	J2SE Internationalization 5.0	J2SE Internationalization 5.0 New	J2SE Internationalization 1.4.2

Administration

Table 126. Supported specifications and APIs for administration. The product supports the specifications or APIs in this table.

Specification or API	Version 8.5	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java EE Application Deployment specification	Java EE Deployment 1.2	Java EE Deployment 1.2	Java EE Deployment 1.2 New	J2EE Deployment 1.1	J2EE Deployment 1.1 New
J2EE Extension Mechanism Architecture	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2	J2EE Extension Mechanism Architecture 1.4.2 New
Java Management Extensions (JMX) JSR-000003	JMX 1.4	JMX 1.4	JMX 1.4	JMX 1.2	JMX 1.2 New
Java Management Extensions (JMX) Remote API	JMX Remote API 1.0	JMX Remote API 1.0	JMX Remote API 1.0	JMX Remote API 1.0 New	Not applicable
Java Virtual Machine (JVM) specification See WebSphere Application Server detailed system requirements.	JVM 6	JVM 6	JVM 6 New	JVM 5.0 New	JVM 1.4.2
Logging API specification (JSR 47)	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0	Logging API specification (JSR 47) 1.0 New

Chapter 45. WebSphere Application Server architecture

This article introduces the parts of the WebSphere Application Server.

Servers

WebSphere Application Server. An application server is a Java virtual machine (JVM) running user applications. Application servers use Java technology to extend web server capabilities to handle web application requests. An application server makes it possible for a server to generate a dynamic, customized response to a client request. The WebSphere Application Server provides application servers.

IBM i For more introduction, refer to “Introduction: Application servers” on page 994.

For more information, refer to Server collection.

Web servers. In the WebSphere Application Server, an application server works with a web server to handle requests for web applications. The application server and web server communicate using an HTTP plug-in for the web server.

For more information, refer to Implementing a web server plug-in.

Resources

JMS providers. The product supports messaging by providing a range of Java Message Service (JMS) providers that conform to the JMS specifications. There are three main types of JMS provider that can be configured in WebSphere Application Server: The WebSphere Application Server default messaging provider (uses service integration as the provider), the WebSphere MQ messaging provider (uses your WebSphere MQ system as the provider) and 3rd party messaging providers (use another company's product as the provider).

For more information, refer to “Introduction: Messaging resources” on page 1046.

Environment

Environment settings help handle requests among web applications, web containers, and the application server.

Virtual hosts. A virtual host is a configuration enabling a single host to resemble multiple logical hosts. Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number that are used to request the servlet, for example: *hostname*:80. The DNS alias might be the host name and port of a web server that routes to the application server or the actual host name and port on which the application server is listening. Java Platform, Enterprise Edition (Java EE) web modules are mapped to a virtual host at installation time. Web modules that use the same virtual host can dispatch to resources within one another.

For more information, refer to Virtual hosts.

WebSphere variables. Variables are used to control settings and properties relating to the server environment. WebSphere variables are used to configure product path names such as JAVA_HOME, and environmental customization values.

For more information, refer to WebSphere variables.

Shared libraries. Shared libraries are files used by multiple applications. You can define a shared library at the node or server level. You can then associate the library to an application or server in order for the classes represented by the shared library to be loaded in either a server-wide or application-specific class loader.

For more information, refer to Managing shared libraries.

System administration

Administrative console. The administrative console is a graphical interface that provides many features to guide you through deployment and systems administration tasks. Use it to explore available management options.

For more introduction, refer to "Introduction: Administrative console" on page 990.

Scripting client (wsadmin). The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. You can also submit scripting language programs to run. The wsadmin tool is intended for production environments and unattended operations.

For more introduction, refer to "Introduction: Administrative scripting (wsadmin)" on page 990.

Administrative programs (Java Management Extensions). The product supports a Java programming interface for developing administrative programs. All of the administrative tools that are supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification.

For more introduction, refer to "Introduction: Administrative programs" on page 991.

Command line tools. Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

For more introduction, refer to "Introduction: Administrative commands" on page 991.

Configuration files. Product configuration data resides in XML files that are manipulated by the previously mentioned administrative clients.

For more introduction, refer to "Introduction: Administrative configuration data" on page 992.

Monitoring and tuning

Monitoring tools. Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environments. Performance monitoring tools include :

- Performance Monitoring Infrastructure (PMI) for monitoring to understand overall system health. For more information, see Performance Monitoring Infrastructure (PMI).
- Request metrics for monitoring to understand resource usage. For more information, see Why use request metrics?.
- Tivoli Performance Viewer (TPV) for viewing the performance data that you collected. For more information, see Why use Tivoli Performance Viewer?.

Tuning tools. Tuning the product helps you obtain the best performance from your website. Tuning the product involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

For more information, refer to Obtaining advice from the advisors.

Troubleshooting

Diagnostic tools. Diagnostic tools help you isolate the source of problems. Many diagnostic tools are available for this product.

IBM i For more information, refer to Working with troubleshooting tools.

Support and self-help IBM Support can assist in deciphering the output of diagnostic tools. Refer to the WebSphere Application Server Technical Support website for current information on known problems and their resolution. Documents at this site can save you time gathering information that is needed to resolve a problem.

For more information, refer to the WebSphere Application Server Support page.

Three-tier architectures

WebSphere Application Server provides the application logic layer in a three-tier architecture, enabling client components to interact with data resources and legacy applications.

Collectively, three-tier architectures are programming models that enable the distribution of application functionality across three independent systems, typically:

- Client components running on local workstations (tier one)
- Processes running on remote servers (tier two)
- A discrete collection of databases, resource managers, and mainframe applications (tier three)

The following diagram outlines the three tier levels. The tiers are logical . They might or might not be running on the same physical server.

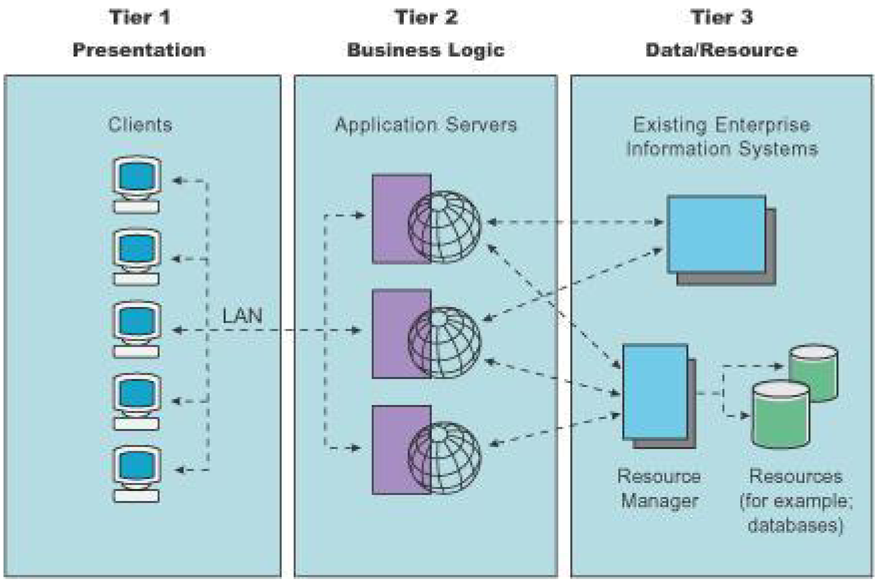


Figure 117. Three tier architecture

First tier. Responsibility for presentation and user interaction resides with the first-tier components. These client components enable the user to interact with the second-tier processes in a secure and intuitive manner. WebSphere Application Server supports several client types. Clients do not access the third-tier services directly. For example, a client component provides a form on which a customer orders products. The client component submits this order to the second-tier processes, which check the product databases and perform tasks that are needed for billing and shipping.

Second tier. The second-tier processes are commonly referred to as the "application logic layer". These processes manage the business logic of the application, and are permitted access to the third-tier services. The application logic layer is where most of the processing work occurs. Multiple client components can access the second-tier processes simultaneously, so this application logic layer must manage its own transactions.

In the previous example, if several customers attempt to place an order for the same item, of which only one remains, the application logic layer must determine who has the right to that item, update the database to reflect the purchase, and inform the other customers that the item is no longer available. Without an application logic layer, client components access the product database directly. The database is required to manage its own connections, typically locking out a record that is being accessed. A lock can occur when an item is placed into a shopping cart, preventing other customers from considering it for purchase. Separating the second and third tiers reduces the load on the third-tier services, supports more effective connection management, and can improve overall network performance.

Third tier. The third-tier services are protected from direct access by the client components residing within a secure network. Interaction must occur through the second-tier processes.

Communication among tiers. All three tiers must communicate with each other. Open, standard protocols and exposed APIs simplify this communication. You can write client components in any programming language, such as Java or C++. These clients run on any operating system, by speaking with the application logic layer. Databases in the third tier can be of any design, if the application layer can query and manipulate them. The key to this architecture is the application logic layer.

Chapter 46. Deprecated, stabilized, and removed features

If you are migrating from an earlier release of WebSphere Application Server, you should be aware of the various features that have been deprecated, stabilized, and removed since Version 6.x.

depfeat: Refer to the tables in "Deprecated features," "Stabilized features" on page 1137, and "Removed features" on page 1138 to learn what has been deprecated, stabilized, and removed.

Deprecated features

If a feature is listed as deprecated, IBM might remove this capability in a subsequent release of the product. Future investment will be focused on the strategic function listed under *Recommended Migration Actions* in "Deprecated features." Typically, a feature is not removed until at least two major releases or three full years (whichever time period is longer) after the release in which that feature is deprecated. For example, features that are deprecated in Version 6.0, Version 6.0.1, or Version 6.0.2 are not removed from the product until after Version 7.0 because both Version 6.0.x and Version 6.1.x are major releases. In rare cases, it might become necessary to remove features sooner; such cases are indicated clearly and explicitly in the descriptions of these deprecated features.

For information on features that have been deprecated in this and earlier releases of WebSphere Application Server, read "Deprecated features."

Stabilized features

If a feature is listed as stabilized, IBM does not currently plan to deprecate or remove this capability in a subsequent release of the product; but future investment will be focused on the alternative function listed under *Strategic Alternative* in "Stabilized features" on page 1137. You do not need to change any of your existing applications and scripts that use a stabilized function; but you should consider using the strategic alternative for new applications.

For information on features that have been stabilized in this release of WebSphere Application Server, read "Stabilized features" on page 1137.

Removed features

For information on features that have been removed in this and earlier releases of WebSphere Application Server, read "Removed features" on page 1138.

Deprecated features

If you are migrating your configuration from an earlier release of WebSphere Application Server, you should be aware of the various features that have been deprecated in this and earlier releases.

If a feature is listed here as deprecated, IBM might remove this capability in a subsequent release of the product. Future investment will be focussed on the strategic function listed under *Recommended Migration Actions*. Typically, a feature is not removed until at least two major releases or three full years (whichever time period is longer) after the release in which that feature is deprecated. For example, features that are deprecated in Version 6.0, Version 6.0.1, or Version 6.0.2 are not removed from the product until Version 7.0 because both Version 6.0.x and Version 6.1.x are major releases. In rare cases, it might become necessary to remove features sooner; such cases are indicated clearly and explicitly in the descriptions of these deprecated features in this article.

The following tables summarize deprecated features by version and release. The tables indicate what is deprecated—such as application programming interfaces (APIs), scripting interfaces, tools, wizards, publicly exposed configuration data, naming identifiers, and constants. Where possible, the tables also indicate the recommended migration action.

This article contains the following deprecation tables:

Features deprecated in Version 8.5

Table 127. Features deprecated in Version 8.5. This table describes the features that are deprecated in Version 8.5.

Category	Deprecation	Recommended Migration Action
Application services	The com.ibm.websphere.management.application.client.EnvEntry_processAnnotations property	Use the com.ibm.websphere.management.application.client.EnvEntry_processBindings property
	The DOJO widgets associated with the Communications Enabled Applications (CEA) feature	Use the CEA DOJO 1.5 widget source and description that are available from the publicly accessible developerWorks article at http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&topic=ceawidgets.
	All application programming interfaces (APIs) related to the Common Event Infrastructure (CEI) module The following files are used by the CEI module and are deprecated: • Files in *app_server_root*/bin/: – eventbucket.sh/eventbucket.bat – eventcatalog.sh/eventcatalog.bat – eventemit.sh/eventemit.bat – eventpurge.sh/eventpurge.bat – eventpurgepool.sh/eventpurgepool.bat – eventquery.sh/eventquery.bat • Files in *app_server_root*/systemApps/: – EventService.ear/META-INF – EventService.ear/DataStoreEjb.jar – EventService.ear/EventCatalogEjb.jar – EventService.ear/EventServerEjb.jar – EventService.ear/EventServerMdb.jar • Files in *app_server_root*/plugins/: – com.ibm.events.client.jar – com.ibm.events.service.jar • File in *app_server_root*/features/: – com.ibm.events.service_6.1.0.0 • Files in *app_server_root*/properties/schemas/: – cei.xsd – ceiservice.xs • Files in *app_server_root*/util/event/: – ceiconfigreload.jacl – eventbucket.jacl – eventcatalog.jacl – eventemit.jacl – eventpurge.jacl – eventpurgepool.jacl – eventquery.jacl • Files in *app_server_root*/web/configDocs/packages/cei/: – classes-frame.html – DataStoreProfile.html – DistributionQueue.html – EmitterFactoryProfile.html – EventBusTransmissionProfile.html – EventGroupProfile.html – EventGroupProfileList.html – EventInfrastructureProvider.html – EventServerProfile.html – FilterFactoryProfile.html – JMSTransmissionProfile.html	For new applications that require event-based models, use Service Integration Bus (SIBus). Design new applications using SIBus-provided APIs for sending and receiving events.
	The com.ibm.portal.* packages: • com.ibm.portal • com.ibm.portal.dynamicui • com.ibm.portal.help • com.ibm.portal.jndi • com.ibm.portal.portlet.service	There is no recommended migration action.

Features deprecated in Version 8.0.0.1

Table 128. Features deprecated in Version 8.0.0.1. This table describes the features that are deprecated in Version 8.0.0.1.

Category	Deprecation	Recommended Migration Action
System administration	Support for the following: • Sybase Adaptive Server Enterprise (ASE) 12.5.X See the End of Life Notice for Adaptive Server Enterprise (ASE) 12.5.X. • Sybase jConnect 5.5 See the End of Life Notice for jConnect Version 5.5.	Move to the following: • Sybase Adaptive Server Enterprise (ASE) 15.X • Sybase jConnect 6.05 or jConnect 7.0

Features deprecated in Version 8.0

IBM i

Table 129. Features deprecated in Version 8.0. This table describes the features that are deprecated in Version 8.0.

Category	Deprecation	Recommended Migration Action
Application services	The following historyInfo utility command-line arguments: • -components • -maintenancePackageID	Be aware of the following when you use the historyInfo utility: • -component Do not use this argument. It now performs no action. • -maintenancePackageID This argument now performs an action that is equivalent to -offeringID. Use -offeringID.
	The following versionInfo utility command-line arguments: • -componentDetail • -components • -maintenancePackageDetail • -maintenancePackages	Be aware of the following when you use the versionInfo utility: • -componentDetail Do not use this argument. It now performs no action. • -components Do not use this argument. It now performs no action. • -maintenancePackageDetail This argument now performs an action that is equivalent to -fixpackDetail plus -ifixDetail. Use -fixpackDetail and -ifixDetail. • -maintenancePackages This argument now performs an action that is equivalent to -fixpacks plus -ifixes. Use -fixpacks and -ifixes.
	The following methods in the com.ibm.websphere.product.WASDirectory class: • public WASComponent getInstalledComponentByName(String componentName) • public WASComponent[] getInstalledComponentList() • public boolean isComponentInstalled(String componentName) • public WASMaintenancePackage[] getHistoryMaintenancePackageList() • public WASMaintenancePackage getInstalledMaintenancePackageByID(String mpID) • public WASMaintenancePackage[] getInstalledMaintenancePackageList() • public boolean isMaintenancePackageInstalled(String ID)	Do not use these methods. • public WASComponent getInstalledComponentByName(String componentName) This method now returns a null object. • public WASComponent[] getInstalledComponentList() This method now returns an empty list. • public boolean isComponentInstalled(String componentName) This method now returns as false. • public WASMaintenancePackage[] getHistoryMaintenancePackageList() This method is replaced by public IMEvent[] getHistoryEventList(). • public WASMaintenancePackage getInstalledMaintenancePackageByID(String mpID) This method is replaced by public IMOffering getInstalledOfferingByID(String productID). • public WASMaintenancePackage[] getInstalledMaintenancePackageList() This method is replaced by public IMOffering[] getInstalledOfferingList(). • public boolean isMaintenancePackageInstalled(String ID) This method is replaced by public boolean isThisProductInstalled(String id).

Table 129. Features deprecated in Version 8.0 (continued). This table describes the features that are deprecated in Version 8.0.

Category	Deprecation	Recommended Migration Action
Application services	The following classes under the com.ibm.websphere.product.* package: • com.ibm.websphere.product.metadata.WASComponent • com.ibm.websphere.product.metadata.WASComponentUpdate • com.ibm.websphere.product.metadata.WASMaintenancePackage • com.ibm.websphere.product.WASProductException • com.ibm.websphere.product.history.WASHistoryException • com.ibm.websphere.product.WASProduct. • com.ibm.websphere.product.history.WASHistory	Use the following guidelines: • com.ibm.websphere.product.metadata.WASComponent Do not use this class. All public methods in this class now return either null objects or empty lists. • com.ibm.websphere.product.metadata.WASComponentUpdate Do not use this class. All public methods in this class now return either null objects or empty lists. • com.ibm.websphere.product.metadata.WASMaintenancePackage Do not use this class. • com.ibm.websphere.product.WASProductException Do not use this class. Use the com.ibm.websphere.product.WASDirectoryException class. • com.ibm.websphere.product.history.WASHistoryException Do not use this class. Use the com.ibm.websphere.product.WASDirectoryException class. • com.ibm.websphere.product.WASProduct. Do not use this class. Use the com.ibm.websphere.product.WASDirectory class. • com.ibm.websphere.product.history.WASHistory Do not use this class. Use the com.ibm.websphere.product.WASDirectory class.
	The following constants under the com.ibm.websphere.product.WASDirectory class: • ID_BASE • ID_EXPRESS • ID_ND The following constants under the com.ibm.websphere.product.WASDirectory and com.ibm.websphere.product.utils.WASDirectoryHelper classes: • ID_PME • ID_WBI • ID_JDK • ID_EMBEDDED_EXPRESS • ID_XD • ID_CLIENT • ID_PLG • ID_IHS • ID_WXD • ID_NDDMZ • ID_UPDI	WASDirectory interfaces are independent of product IDs and treat them as a string. Any product-specific callers should specify product IDs by themselves.
	Support for Java API for XML Registries (JAXR)	Use UDDI Version 3.
Security	com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus Tivoli Access Manager (TAM) trust association interceptor (TAI) interface	Get the latest version of Tivoli code from the Tivoli Access Manager Trust Association Interceptor Plus website.

Table 129. Features deprecated in Version 8.0 (continued). This table describes the features that are deprecated in Version 8.0.

Category	Deprecation	Recommended Migration Action
System administration	`enableJvm` command	Use the `managesdk` command.
	`createServerType` command in the ServerManagement command group for the AdminTask object	Do not create new server types.
	The following commands in the ManagedNodeGroup command group for the AdminTask object: • createManagedNodeGroup • deleteManagedNodeGroup • addMemberToManagedNodeGroup • deleteMemberFromManagedNodeGroup • queryManagedNodeGroups • getManagedNodeGroupMembers • getManagedNodeGroupInfo • modifyManagedNodeGroupInfo The following commands in the JobManagerNode command group for the AdminTask object: • cleanupManagedNode • queryManagedNodes • getManagedNodeProperties • modifyManagedNodeProperties • getManagedNodeKeys	Use the following commands in the TargetGroup command group for the AdminTask object: • createTargetGroup • deleteTargetGroup • addMemberToTargetGroup • deleteMemberFromTargetGroup • queryTargetGroups • getTargetGroupMembers • getTargetGroupInfo • modifyTargetGroupInfo Use the following commands in the JobManagerNode command group for the AdminTask object: • cleanupTarget • queryTargets • getTargetProperties • modifyTargetProperties • getTargetKeys
	Service log, commonly named `activity.log`	Use one of the following to access log content: • `SystemOut.log` file when your system is configured to use basic log and trace mode • High Performance Extensible Logging (HPEL) LogViewer command when your system is configured to use HPEL log and trace mode Configure your servers to use HPEL log and trace mode and use the HPEL API if you need to be able to merge log file content from multiple servers. Use the HPEL log and trace mode and use the HPEL LogViewer command if you need to be able to render log content in Common Base Event XML format.
	Support for deploying Java Platform, Enterprise Edition (Java EE) modules or applications on an application server using the Java EE Application Deployment API specification JSR-88	Use any of a number of other options to deploy applications to the server, including wsadmin scripting and JMX MBeans. The closest method to using the Java EE Deployment API would be using WebSphere JMX MBeans. Read Ways to install enterprise applications or modules for more information.
	DataPower® appliance manager	You can continue to use the DataPower appliance manager to manage existing supported DataPower appliances until it is removed from the product or until the appliances are out of service. In WebSphere Application Server Version 8.0, the following appliances are supported: • 9001 • 9002 • 9003/7993 • 9004/9235 – XS40, XI50, XB60, XM70 • 9004/9235 – XA35, XM70FC The end-of-service dates for the appliances are documented inIBM WebSphere DataPower SOA Appliances End of Service dates. New appliances not on the previous list are managed through a separate DataPower appliance management offering that is also capable of managing existing appliances.
Tivoli Performance Viewer	Use of the Scalable Vector Graphics (SVG) format by the Tivoli Performance Viewer to plot graphs	No action is required. The Tivoli Performance Viewer now uses the Dojo format to plot graphs. **Note:** To go back to the earlier style of graph, which supports SVG and image formats, set the com.ibm.websphere.tpv.DojoGraph JVM system property to false.

Features deprecated in Version 7.0

IBM i

Table 130. Features deprecated in Version 7.0. This table describes the features that are deprecated in Version 7.0.

Category	Deprecation	Recommended Migration Action
Application programming model	registerSynchronizationCallbackForCurrentTran method in the com.ibm.websphere.jtaextensions.ExtendedJTATransaction interface	Use the registerInterposedSynchronization method of the TransactionSynchronizationRegistry interface instead.
	com.ibm.ws.extensionhelper.TransactionControl interface	Use the com.ibm.wsspi.uow.UOWManager interface instead.
	HttpServletRequestProxy class in the com.ibm.websphere.servlet.request package	Use the HttpServletRequestWrapper class instead of the HttpServletRequestProxy class. You can use the subclasses of this class to overload or enhance the functionality of a server-provided HttpServletRequest.
	HttpServletResponseProxy class in the com.ibm.websphere.servlet.response package	Use the HttpServletResponseWrapper class instead of the HttpServletResponseProxy class. You can use the subclasses of this class to overload or enhance the functionality of a server-provided HttpServletResponse.
	The following classes, interfaces, methods, and fields of the WebSphere relational resource adapter: • Classes: – com.ibm.websphere.rsadapter.JdbcAccessorImpl – com.ibm.websphere.rsadapter.OracleDataStoreHelper • Interfaces: – com.ibm.websphere.rsadapter.Beginnable – com.ibm.websphere.rsadapter.HandleStates – com.ibm.websphere.rsadapter.Reassociateable – com.ibm.websphere.rsadapter.WSNativeConnectionAccessor • Methods: – com.ibm.websphere.rsadapter.WSCallHelper.setConnectionError (Object conn) – com.ibm.websphere.rsadapter.WSCallHelper.call – com.ibm.websphere.rsadapter.WSConnection.getClientInformation – com.ibm.websphere.rsadapter.WSConnection.setClientInformation – com.ibm.ws.rsadapter.cci.WSResourceAdapterBase.getNativeConnection (javax.resource.cci.Connection) – com.ibm.ws.rsadapter.cci.WSResourceAdapterBase.getNativeConnection (com.ibm.ws.rsadapter.jdbc.WSJdbcConnection) – com.ibm.ws.rsadapter.jdbc.WSJdbcUtil.getNativeConnection (com.ibm.ws.rsadapter.jdbc.WSJdbcConnection) • Fields: – com.ibm.websphere.rsadapter.DataStoreHelper.ORACLE_HELPER – com.ibm.websphere.rsadapter.WSConnection.CLIENT_ACCOUNTING_INFO – com.ibm.websphere.rsadapter.WSConnection.CLIENT_APPLICATION_NAME – com.ibm.websphere.rsadapter.WSConnection.CLIENT_ID – com.ibm.websphere.rsadapter.WSConnection.CLIENT_LOCATION – com.ibm.websphere.rsadapter.WSConnection.CLIENT_OTHER_INFO – com.ibm.websphere.rsadapter.WSConnection.CLIENT_TYPE	If you are using the OracleDataStoreHelper, switch to the Oracle 11g JDBC driver and use the Oracle11gDataStoreHelper instead. Instead of using getNativeConnection, use the Java Database Connectivity (JDBC) 4.0 wrapper pattern. Instead of WSConnection client information, use JDBC 4.0 client-information APIs. Instead of com.ibm.websphere.rsadapter.WSCallHelper.setConnectionError (Object conn), use the com.ibm.websphere.rsadapter.WSCallHelper.setConnectionError (Object conn, boolean logEvent) method. The new method provides a boolean parameter that allows you to control whether or not connection error events are logged to system out. Instead of WSCallHelper.call, use standard Java reflections APIs. If you are using the ORACLE_HELPER, switch to the Oracle 11g JDBC driver and use the ORACLE_11G_HELPER instead.

Table 130. Features deprecated in Version 7.0 (continued). This table describes the features that are deprecated in Version 7.0.

Category	Deprecation	Recommended Migration Action
Application programming model	The following session-management function: • Global session sharing through the Servlet21SessionCompatibility property • Support for any session-manager properties as system properties • Support for any session-manager properties as web container custom properties	For session sharing, redesign your applications so that the session is appropriately scoped at the web module as specified in the Java Servlet Specification Version 2.2 and later. If data must be shared across the Web-module boundary, use the Websphere Extension, shared session context = true, or use the IBMApplicationSession API. Rather than specifying session manager properties as system or web container custom properties, use session-manager custom properties.
	Connection validation by SQL query	Use the timeout-based validation introduced with JDBC 4.0.
	Classes: • SERV1\ws\code\admin.thinclient\build\classes\com\ibm\ws\management\ cmdframework\impl\ RemoteCommandMgr*.class • SERV1\ws\code\admin.thinclient\build\classes\com\ibm\ws\management\ cmdframework\impl\ RemoteCommandMgrImpl*.class Interface: SERV1\ws\code\admin.thinclient\src\com\ibm\ws\management\ cmdframework\impl\ RemoteCommandMgr.java Methods: All methods in the RemoteCommandMgr interface and MBean xml, SERV1\ws\code\admin.jmx\src\com\ ibm\ws\management\ descriptor\xml\RemoteCommandMgr.xml Constructor: RemoteCommandMgrImpl()	No migration action is necessary.
	The following proprietary classes that are used to represent and manipulate WS-Addressing endpoint references in Java API for XML Web Services (JAX-WS) 2.0: • com.ibm.websphere.wsaddressing.jaxws.W3CEndpointReference • com.ibm.websphere.wsaddressing.jaxws.SubmissionEndpointReference • com.ibm.websphere.wsaddressing.jaxws.EndpointReferenceConverter	Use the following classes instead: • javax.xml.ws.wsaddressing.W3CEndpointReference • com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReference • com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceConverter
	The following WebSphere Common Configuration Model (WCCM) types: • SIBJMSProvider • SIBJMSConnectionFactory • SIBJMSQueueConnectionFactory • SIBJMSTopicConnectionFactory • SIBJMSQueue • SIBJMSTopic	If one of your Jython or Jacl wsadmin scripts uses any of these types, modify the script to use the correct AdminTask command to complete the equivalent function. For example: `AdminTask.listSIBJMSQueues()`
Environment	IBM HTTP Server (IHS) mod_file_cache module	Migrate your IHS configuration directives from mod_file_cache to the appropriate provided cache mechanism—either mod_mem_cache or mod_cache.
	IHS mod_ibm_ldap module	Migrate your mod_ibm_ldap configuration directives to the equivalent mod_ldap directives.
	IHS mod_mime_magic module	Remove any IHS configuration directives associated with this module.
	IHS mod_proxy_ftp module	Remove any IHS configuration directives associated with this module.
	IHS mod_afpa_cache module Adaptive Fast Path Architecture (AFPA) is being deprecated for both AIX® and Windows operating systems for both static and dynamically generated content caching.	Remove any IHS configuration directives associated with this module.
Installation and maintenance tools	Support for using IBM Update Installer for WebSphere Software Version 6.x to apply maintenance on WebSphere Application Server Version 6.0.2.21+ and Version 6.1.0.x	Use IBM Update Installer for WebSphere Software Version 7.0. Update Installer Version 7.0 supports applying maintenance on WebSphere Application Server Version 6.0.2.21+, Version 6.1.0.x, and Version 7.0.
	Support for using IBM WebSphere Installation Factory Version 6.1 to create installation packages for WebSphere Application Server Version 6.1	Use IBM WebSphere Installation Factory Version 7.0. WebSphere Installation Factory Version 7.0 supports creating installation packages for WebSphere Application Server Version 6.1 and Version 7.0.
J2EE resources	The following features: • Support for using JMS providers that are not compliant with the J2EE Connector Architecture 1.5 specification is deprecated. The WebSphere Application Server Version 5 default messaging provider was deprecated in Version 6.1. • The WebSphere MQ messaging provider was updated in Version 7.0 to support the J2EE Connector Architecture 1.5. Support for other usages of this provider are deprecated.	Use JMS providers that are compliant with the J2EE Connector Architecture 1.5 specification.

Table 130. Features deprecated in Version 7.0 (continued). This table describes the features that are deprecated in Version 7.0.

Category	Deprecation	Recommended Migration Action
Data access	disablePK54589 system property	Configure the data source custom property isConnectionSharingBasedOnCurrentState. If you are using disablePK54589=true, you can replace it with isConnectionSharingBasedOnCurrentState=false.
Programming	Shipment of JavaServer Faces widget library (JWL) with WebSphere Application Server	Manually create a shared library for JWL using the jar from Rational Application Developer.
Servers	Shipment of Apache Struts 1.1, 1.2.4, and 1.2.7 as optional libraries within WebSphere Application Server	If you would like to continue using these versions of Apache Struts, they are available from the Apache Struts website.
System administration	Option to install the Pluggable application client feature for the IBM Application Client for WebSphere Application Server	Use the new EJB Thin Application Client feature instead.
	The following service integration bus (SIBus) security features: • -secure flag on the **`createSIBus`** and the **`modifySIBus`** commands • **`listInheritSenderForTopic`**, **`listInheritReceiverForTopic`**, and **`listInheritDefaultsForDestination`** commands • Inter-engine authentication alias	Perform the following actions: • Use the -busSecurity flag instead of the -secure flag. • Replace usages of the **`listInheritSenderForTopic`**, **`listInheritReceiverForTopic`**, and **`listInheritDefaultsForDestination`** commands with the **`isInheritSenderForTopic, isInheritReceiverForTopic`**, and **`isInheritDefaultsForDestinaton`** commands respectively. • Remove any usage of the -interEngineAuthenticationAlias option on the **`createSIBus`** and **`modifySIBus`** commands.
	Collector tool (`collector.bat` or `collector.sh`) that gathers information about the WebSphere Application Server installation and packages it in a Java archive (JAR) file that you can send to IBM Software Support	Use AutoPD.
	Protocol-based proxy server templates	Use the administrative console or the wsadmin commands in the ServerManagement command group to select one or multiple protocols for proxy servers. For more information, read ServerManagement command group for the AdminTask object
	WebSphere Touchpoint (WAS.admin.wstp component—all classes and methods)	Use the other standard management interfaces in WebSphere Application Server.
	Commands in the SecureConversation command group for the AdminTask object	Use the commands in the WSSCacheManagement command group to manage Web Services Security (WS-Security) distributed-cache configurations.
	The following Lightweight Directory Access Protocol (LDAP) configuration names in the virtual member manager (VMM) federated repository: • SECUREWAY, IDS4, IDS51, and IDS6 • DOMINO5, DOMINO6, and DOMINO65 • AD2000 and AD2003	Use the following configuration names: • IDS rather than SECUREWAY, IDS4, IDS51, or IDS6 • DOMINO rather than DOMINO5, DOMINO6, or DOMINO65 • AD rather than AD2000 or AD2003
Web services	Support for the '2006/02' WS-Addressing Web Services Description Language (WSDL) binding namespace	Replace any uses of the '2006/02' namespace in WSDL files with uses of the '2006/05' namespace
	Web Services Distributed Management (WSDM) interface	Use the other standard management interfaces in WebSphere Application Server.
	IBM proprietary SOAP over Java Message Service (JMS) protocol for Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) applications	Use the standard SOAP over JMS protocol. For more information, read SOAP over JMS protocol and SOAP over Java Message Service . **Restriction:** If your client application invokes enterprise-bean based web services that are supported by a release of WebSphere Application Server that is earlier than Version 7.0, you must continue to use the IBM proprietary SOAP over JMS protocol to access those web services.

Features deprecated in Version 6.1

IBM i

Table 131. Features deprecated in Version 6.1. This table describes the features that are deprecated in Version 6.1.

Category	Deprecation	Recommended Migration Action

Application programming model	setDatabaseDefaultIsolationLevel(int) method in the com.ibm.websphere.rsadapter.DataStoreHelperMetaData class	Start using the following instead:
		`public final void`
		`setDatabaseDefaultIsolationLevel`
		`(int helperDefaultLevel,`
		`int cusDefinedWasDefaultIsoLevel)`

	The following class and interface in the Mediation Framework runtime: • com.ibm.websphere.sib.mediation.handler.SIMessageContextException class • com.ibm.websphere.sib.mediation.messagecontext.SIMediationBean MessageContext interface	Replace all uses of the com.ibm.websphere.sib,.mediation.handler.SIMessageContextException class with the com.ibm.websphere.sib.mediation.handler.MessageContextException class. Replace all uses of the com.ibm.websphere.sib.mediation.messagecontext.SIMediationBean MessageContext interface with an equivalent interface. WebSphere Application Server does not provide an implementation of this interface.
	The following web container message bean functions: • startTransports • stopTransports • restartWebApplication	Begin moving to the channel framework. The channel framework provides the TransportChannelService message bean, which is more flexible and has more methods than the current web container transport-related methods.
	Support for deploying container-managed entity beans to a generic SQL database	If an application uses SQL92 or SQL99 because the application has to run with different relational databases, use the IBM tooling to generate deployed code for each database vendor or version that the application might use. At installation time, specify the database vendor or version that will used with WebSphere Application Server.
	IBM WebSphere Studio tools runtime support provided by the following classes (which were used to leverage Visual Age for Java tooling): • com.ibm.webtools.runtime.AbstractStudioServlet • com.ibm.webtools.runtime.BuildNumber • com.ibm.webtools.runtime.NoDataException • com.ibm.webtools.runtime.StudioPervasiveServlet • com.ibm.webtools.runtime.TransactionFailureException • com.ibm.webtools.runtime.WSUtilities	Rearchitect your applications to use standard J2EE coding conventions.
	CUSTOM_HELPER constant field in the com.ibm.websphere.rsadapter.DataStoreHelper class API	If you create your own DataStoreHelper implementation class, do not invoke setHelperType(DataStoreHelper.CUSTOM_HELPER). Instead, let the HelperType value be set by the implementation class from which it inherits.
J2EE resources	Support for the ability to connect from either an application server or a J2EE application client to the JMS Server component of the embedded messaging feature in WebSphere Application Server Version 5 This deprecation includes the following: • Ability to define JMS resource definitions for the Version 5 default messaging provider • Ability to establish connections from client applications that are either running in a Version 5 environment or utilizing Version 5 default messaging provider resource definitions	Perform the following actions: 1. Ensure that any JMS Server messaging providers that are hosted on WebSphere Application Server Version 5.1 application servers are moved onto Version 6.0 or later application servers. This task is handled automatically when you migrate a Version 5.x server to Version 6.0 or later. 2. Change all JMS resource definitions to use the new Version 6 default messaging provider instead of the Version 5 default messaging provider.

Table 131. Features deprecated in Version 6.1 (continued). This table describes the features that are deprecated in Version 6.1.

Category	Deprecation	Recommended Migration Action
System administration	`clientUpgrade` command	No migration action is necessary.
	`wasprofile` command	Use the `manageprofiles` command.
	The following `WASPostUpgrade` command parameters: • -import *xmi_data_file* • -substitute "*key1=value1*[;*key2=value2*;[...]]"	No migration action is necessary.
	Cloudscape datastore helper (com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper) and Cloudscape Network Server datastore helper (com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper) as well as their types in DataStoreHelper	For existing configurations, no migration action is necessary. The migration utility changes the deprecated Cloudscape helpers to Derby helpers. For new configurations, use the Derby datastore helpers and types instead of the Cloudscape datastore helpers.
	DB2 Legacy CLI-based Type 2 JDBC Driver provider	Start using the DB2 Universal JDBC Driver Provider.
	Logical pool distribution support (com.ibm.websphere.csi.ThreadPoolStrategy.LogicalPoolDistribution)	No migration action is necessary. When this function is removed, however, all custom Object Request Broker (ORB) properties that you specified for it will be ignored. The custom ORB properties of interest are com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.*.
	ORB thread pool configuration as part of the Server object in the `server.xml` file	Use the thread pool configuration that is part of the ServerIndex object in the `serverindex.xml` file.
	JVM system property com.ibm.websphere.sendredirect.compatibility	Begin modifying your applications to redirect non-relative URLs, those starting with a forward slash ("/"), relative to the servlet container root (*web_server_root*) instead of the web application context root. Read the *Java Servlet 2.4 Specification*, which is available for download at http://jcp.org/aboutJava/communityprocess/final/jsr154/, for information on how sendRedirect should behave.
	Web container PageList Servlet custom extension, including the following classes: • com.ibm.servlet.ClientList • com.ibm.servlet.ClientListElement • com.ibm.servlet.MLNotFoundException • com.ibm.servlet.PageListServlet • com.ibm.servlet.PageNotFoundException	Rearchitect your applications to use javax.servlet.filter classes rather than com.ibm.servlet classes. Starting with the Java Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. They also allow provide chaining functionality as well as functionality for embellishing or truncating responses.
	The following custom properties for a data source: • validateNewConnection • validateNewConnectionRetryCount • validateNewConnectionRetryInterval	The product now offers these properties as pre-configured options, which are the replacement properties in the following list. To avoid runtime error messages, permanently disable the original custom properties by deleting them from the list of custom properties. • validateNewConnection is replaced by **Pretest new connection** • validateNewConnectionRetryCount is replaced by **Number of retries** • validateNewConnectionRetryInterval is replaced by **Retry interval** **Note:** If the new properties and old properties coexist, the new properties take precedence.
	Simple WebSphere Authentication Mechanism (SWAM)	Use the Lightweight Third-Party Authentication (LTPA) mechanism.
Security	LoginHelper CORBA authentication helper function (com.ibm.ws.security.util.LoginHelper)	Migrate to the Java Authentication and Authorization Service (JAAS) programming model. For information on this migration, read Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS).
	com.ibm.ws.security.web.WebSealTrustAssociationInterceptor trust association interceptor (TAI) interface This Tivoli TAI interceptor that implements the WebSphere Application Server TAI interface was provided to support WebSEAL Version 4.1.	If you plan to use WebSEAL 5.1 or later, you should migrate to use the com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus interceptor.
	Support for the Java Virtual Machine Profiler Interface (JVMPI) is deprecated along with the following related JVM runtime counters: • ObjectMovedCount • ObjectFreedCount • ObjectAllocateCount	Begin moving to the Java Virtual Machine Tool Interface (JVMTI). For more information, read JVM Tool Interface (JVMTI).
Performance	Support for the Java Virtual Machine Debugger Interface (JVMDI)	Begin moving to the Java Virtual Machine Tool Interface (JVMTI). For more information, read JVM Tool Interface (JVMTI).
	Message ID format that is used in WebSphere Application Server Version 6.0.x and earlier The message prefixes for log files were not previously registered with the primary message registry. WebSphere Application Server Version 6.1.x and later use compliant message prefixes in the output logs.	Use the `convertlog` command and the MessageConverter class to assist you in migrating tools that rely on the older message format.

Table 131. Features deprecated in Version 6.1 (continued). This table describes the features that are deprecated in Version 6.1.

Category	Deprecation	Recommended Migration Action
Problem determination	The com.ibm.etools.logging.util plug-in, the `logutil.jar` file • Logging facility Logging facility used for logging Java primitives and complex objects to named loggers; configurable with predefined filtering levels, Logging Agent and file sinks, and output formats through an API, Eclipse plug-in manifest, or Eclipse preference panel • Logging agent XML-based messaging agent used in conjunction with the IBM Agent Controller to write log and trace XML records to a logging service remotely attachable through an API or Test and Performance Tools Platform (TPTP), formally Hyades, Eclipse workbench • Problem determination artifacts and messages Original implementation of the Manageability (M12) Model Problem Determination Architecture Version 1.5 and Problem Determination Artifacts Common Data Model specification used for capturing and encoding log and trace data • Distributed correlator service (DCS) Distributed correlator service that is used for instrumenting correlation identifiers for correlating log and trace data across one or more hosts • Java client bindings Java client bindings used to communicate with the IBM Agent Controller to launch local and remote processes, attach to running processes, and monitor active agents in a secure client environment	Begin moving plug-ins and application code using configuration files, classes, methods, or variables in the com.ibm.etools.logging.util plug-in to the following replacements: • Logging facility **Replacement:** Java Logging APIs in Java Version 1.4.0+; Logging Agent support for the Java Logging APIs provided in TPTP and Common Logging (com.ibm.etools.common.logging/logging.jar) • Logging agent **Replacement:** TPTP Logging Agent (org.eclipse.hyades.logging.core/hlcore.jar) • Problem determination artifacts and messages **Replacement:** Common Base Event Version 1.0.1 specification and TPTP implementation (org.eclipse.hyades.logging.core/hlcbe101.jar) • Distributed correlator service (DCS) **Replacement:** TPTP Correlation Service (org.eclipse.hyades.execution.correlation/hcorrelation.jar) • Java client bindings **Replacement:** TPTP Java Client Bindings (org.eclipse.hyades.execution/hexl.jar) For more information, read the com.ibm.etools.logging.util\doc\IBM_Logging_Utilities_Migration_Guide.html document.

Features deprecated in Version 6.0.2

IBM i

Table 132. Features deprecated in Version 6.0.2. This table describes the features that are deprecated in Version 6.0.2.

Category	Deprecation	Recommended Migration Action

Application programming model	The following methods from class com.ibm.websphere.runtime.ServerName:
	`initialize(java.lang.String*!ENTITY!*cell,`
	`  java.lang.String node,`
	`  java.lang.String server)`
	`was390Initialize(byte[] a_stoken,`
	`  String a_printable_stoken,`
	`  String a_jsabpref,`
	`  int a_pid,`
	`  int an_asid,`
	`  String a_jsabjbnm)`
	`was390Initialize(byte[] a_stoken,`
	`  java.lang.String a_printable_stoken,`
	`  java.lang.String a_jsabpref,`
	`  int a_pid,`
	`  int an_asid,`
	`  java.lang.String a_jsabjbnm,`
	`  java.lang.String a_smcasid)`
		These methods are for WebSphere Application Server runtime use only. Applications should not call these methods.

	Support for HTTP transport configuration	Begin moving to channel-based transport.
Performance	com.ibm.websphere.cache.DistributedLockingMap interface	Do not use the com.ibm.websphere.cache.DistributedLockingMap interface because this interface is not supported by the WebSphere Application Server runtime.
	TYPE_DISTRIBUTED_LOCKING_MAP constant that is defined in the com.ibm.websphere.cache.DistributedObjectCache class	Do not use the TYPE_DISTRIBUTED_LOCKING_MAP constant that is defined in the com.ibm.websphere.cache.DistributedObjectCache class because this constant is not supported by the WebSphere Application Server runtime.
System Administration	The following custom properties for a data source: • dbFailOverEnabled • connRetriesDuringDBFailover • connRetryIntervalDuringDBFailover	Replace the properties with the following: • Use validateNewConnection instead of dbFailOverEnabled. • Use validateNewConnectionRetryCount instead of connRetriesDuringDBFailover. • Use validateNewConnectionRetryInterval instead of connRetryIntervalDuringDBFailover. **Note:** If the new properties and old properties coexist, the new properties take precedence.

Features deprecated in Version 6.0

IBM i

Table 133. Features deprecated in Version 6.0. This table describes the features that are deprecated in Version 6.0.

Category	Deprecation	Recommended Migration Action

Application programming model and container support	Support for the following `tsx` tags in the JavaServer Pages (JSP) engine:
	• repeat
	• dbconnect
	• dbquery
	• getProperty
	• userid
	• passwd
	• dbmodify
		Instead of using the `tsx` tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL). JSTL is supported in WebSphere Application Server Version 6.0, and the tag library is included with the product. Use this table as a guideline for converting `tsx` tags to JSTL tags:
		`tsx tag` — `JSTL tag`
		`tsx:repeat` — `c:forEach`
		`tsx:dbconnect` — `sql:setDataSource`
		`tsx:dbquery` — `sql:query`
		`tsx:getProperty` — `Use standard EL syntax; c:out value="${book.title}" for example, where book is the current index in the result set`
		`tsx:userid` — `Use the user attribute of the setDataSource tag`
		`tsx:passwd` — `Use the password attribute of the setDataSource tag`
		`tsx:dbmodify` — `sql:update`

	The following backend IDs: • SQL92 (1992 SQL Standard) • SQL99 (1999 SQL Standard)	Use other backend IDs.
Application services	JRas Extensions API No further enhancements are planned for JRas support.	Use the equivalent function in the java.util.logging package (JSR47).
	Universal Description, Discovery and Integration (UDDI) Version 2 EJB interface to the UDDI Registry	There is no replacement for the EJB interface. This interface is included in WebSphere Application Server Version 6.0 for compatibility with Version 5.x. Users do not need to take any specific actions and can continue to use the Version 2 EJB API; however, they should be aware that it does not include any UDDI functionality that is new to UDDI Version 3 and the interface might be removed in a subsequent release of WebSphere Application Server.
	The UDDI4J Version 2 class library, the `uddi4jv2.jar` file	Start using the Version 3 UDDI APIs. A client library is provided to simplify constructing and sending UDDI Version 3 requests from Java. This is the IBM UDDI Version 3 Client for Java, provided in `uddiv3client.jar`. The UDDI4J APIs can still be used; however, you should be aware that they do not provide access to any of the new UDDI Version 3 functionality and they might be removed in a subsequent release of WebSphere Application Server.
	All of the low-level UDDI Utility Tools (UUT) APIs, such as BusinessStub, ServiceStub, and so on, These APIs are all replaced with the high-level PromoterAPI interface in the com.ibm.uddi.promoter package.	Start using the PromoterAPI interface in the com.ibm.uddi.promoter package in place of these low-level APIs, which will be removed in a subsequent release of WebSphere Application Server. The PromoterAPI provides the same functionality at a higher level of abstraction.
	The following methods in the J2EE Connector Architecture runtime: • com.ibm.ws.management.descriptor.xml.ConnectionFactory.xml (getPoolContents and getAllPoolContents methods) • com.ibm.websphere.j2c.ConnectionManager interface • com.ibm.websphere.j2c.ConnectionEventListener interface	The methods are replaced as follows: • getPoolContents and getAllPoolContents are replaced with showPoolContents and showAllPoolContents. • ConnectionManager interface is replaced with J2EE Connector Architecture 1.5 LazyAssociatableConnectionManager interface. • ConnectionEventListener interface is replaced with J2EE Connector Architecture 1.5 LazyEnlistableConnectionManager interface. For container-managed authentication aliases, specify the container-managed credentials in the resource binding information for the application.
	ApplicationProfile property on the Work manager panel in the administrative console	Read the information center for the differences between application profiling in Version 5.x and Version 6.0.x.
	The following two items from the Data source panel in the administrative console: • Container-Managed Authentication Alias • DefaultPrincipleMapping	Define the Container-Managed Authentication Alias and DefaultPrincipleMapping properties on the Resource Reference.
	All classes in the com.ibm.websphere.servlet.filter package, including the following: • ChainedRequest • ChainedResponse • ChainerServlet • ServletChain	Rearchitect your applications to use javax.servlet.filter classes rather than com.ibm.websphere.servlet.filter classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. They also allow you to achieve chaining functionality, as well as embellishing and truncating responses.

Table 133. Features deprecated in Version 6.0 (continued). This table describes the features that are deprecated in Version 6.0.

Category	Deprecation	Recommended Migration Action
Application services	Multipurpose Internet Mail Extensions (MIME) filtering MIME filters were first supported in WebSphere Application Server Version 3.5 as a way for servlets to embellish, truncate, and modify the responses generated by other servlets, based on the MIME types of the output content.	The javax.servlet.filters, which were introduced in the Servlet 2.3 specification, allow users to plug in filters that can intercept requests to and responses from servlets. They also have the capability to modify content flowing in either direction. The javax.servlet.filters maintain all of the functionality of MIME filters. javax.servlet.filters are standard APIs, and are supported by all compliant application servers. Refer to the Servlet 2.3 specification for more information.
	Container-managed persistence (CMP) entity beans configured with method level access intent might run into data access problems, like deadlock. Therefore, the method level access intent is deprecated.	Reconfigure CMP entity beans to use bean level access intent, or reconfigure application profiles with WebSphere Application Server Toolkit.
	All of the methods and fields in the com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo classes Therefore, the following methods from com.ibm.websphere.product.WASProduct class (which involves com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo objects) are deprecated: • public product getProductByFilename(String basename) • public product getProductById(String id) • public boolean productPresent(String id) • public boolean addProduct(product aProduct) • public boolean removeProduct(product aProduct) • public Iterator getProducts() • public Iterator getProductNames() • public String loadVersionInfoAsXMLString(String filename) • public String getProductDirName() • public static String computeProductDirName()	Use the following supported methods from com.ibm.websphere.product.WASDirectory: • public WASProductInfo getWASProductInfo(String id) • public boolean isThisProductInstalled(String id) • public WASProductInfo[] getWASProductInfoInstances() • public String getWasLocation() Also, instead of getting product information (name, version, build level, build date) from the old WASProduct API (com.ibm.websphere.product.WASProduct), you should now use the following methods in the WASDirectory class to get that information: • com.ibm.websphere.product.WASDirectory.getName(String) • com.ibm.websphere.product.WASDirectory.getVersion(String) • com.ibm.websphere.product.WASDirectory.getBuildLevel(String) • com.ibm.websphere.product.WASDirectory.getBuildDate(String)
	Data access beans, which are included with WebSphere Application Server in the `databeans.jar` file	Instead of using data access beans, you should use Service Data Objects (SDO). Read Service Data Objects for additional details.
	reloadInterval and reloadingEnabled attributes of the IBM deployment descriptor extensions, including both the WAR file extension (WEB-INF/ibm-web-ext.xmi) and the application extension (META-INF/ibm-application-ext.xmi).	Instead of using deployment descriptor extensions, you should use the reload enable and interval options provided during application deployment.
	com.ibm.websphere.servlet.session.UserTransactionWrapper API	There is no replacement for this API. The UserTransaction object can be placed directly into the HTTP session without using a wrapper.

Table 133. Features deprecated in Version 6.0 (continued). This table describes the features that are deprecated in Version 6.0.

Category	Deprecation	Recommended Migration Action
Security	SOAP-Security (XML digital signature) based on Apache SOAP implementation	Instead of using SOAP-Security, you should migrate your application to JSR-109 implementation of web service. Also, migrate (reconfigure your application) to use WSS (Web Services Security) 1.0 implementation.
	Web Service Security (WSS) draft 13 specification-level support	Applications should be migrated to the supported WSS 1.0 standard. The draft-level support does not provide interoperability with some third-party vendors, as the message level has been changed between the draft and the WSS 1.0 implementation. WSS 1.0 is only supported in J2EE 1.4 applications. Therefore, you need to migrate applications to J2EE 1.4 first. The next step is to use Application Server Toolkit or Rational Application Developer tooling to reconfigure WSS for the migrated application. There is no automatic migration of WSS in this release of Application Server Toolkit or Rational Application Developer tooling for Version 6.0; the migration has to be done manually. The following SPI has also been deprecated: com.ibm.wsspi.wssecurity.config.KeyLocator You need to migrate your implementation to the new SPI for WSS 1.0 support in Version 6.0: com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Finally, the Java Authentication and Authorization Service (JAAS) LoginModule implementation needs to be migrated to the new programming model for JAAS LoginModule in Version 6.0.
	Secure Authentication Service (SAS) IIOP security protocol	Use the Common Secure Interoperability Version 2 (CSIv2) protocols.
	Secure Authentication Service (SAS) CORBA security programming APIs	Migrate from the SAS programming APIs to the Java Authentication and Authorization Service (JAAS). For information on this migration, read Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS).
depl.extension.reg and installdir options for the `install` command in the AdminApp scripting object	There is no replacement for the depl.extension.reg option. In Version 5.x, this option was a no-op. For the installdir option, use the installed.ear.destination option instead.	
Performance	PMI Client API, which was introduced in Version 4.0 to programmatically gather performance data from WebSphere Application Server	The Java Management Extension (JMX) interface, which is part of the J2EE specification, is the recommended way to gather WebSphere Application Server performance data. PMI data can be gathered from the J2EE-managed object message beans, or from the WebSphere PMI Perf message bean. While the J2EE message beans provide performance data about a specific component, the Perf message bean acts as a gateway to the WebSphere Application Server PMI service, and provides access to the performance data for all of the components.

Features deprecated in Version 5.1.1

Table 134. Features deprecated in Version 5.1.1. This table describes the features that are deprecated in Version 5.1.1.

Category	Deprecation	Recommended Migration Action
Application programming model and container support	Web services gateway customization API	Plan over time to replace your existing filters with a combination of JAX-RPC handlers and service integration bus mediations.
Application services	The following Java Database Connectivity (JDBC) drivers: • Microsoft SQL Server 2000 Driver for JDBC • SequeLink JDBC driver for Microsoft SQL Server	If you are using either of these JDBC drivers and still want to use Microsoft SQL Server as your database, switch to the Connect JDBC driver. You can purchase the Connect JDBC driver from DataDirect Technologies.

Features deprecated in Version 5.1

Table 135. Features deprecated in Version 5.1. This table describes the features that are deprecated in Version 5.1.

Category	Deprecation	Recommended Migration Action
Installation and migration tools	The Application Assembly Tool that is used for developing J2EE applications is replaced with the Assembly Tool component of the Application Server Toolkit.	Instead of running the Application Assembly Tool, users will install and run the Assembly Toolkit component of the Application Server Toolkit. The Application Server Toolkit is based on the Eclipse framework. On starting the Application Server Toolkit, the J2EE function is found by opening the J2EE Perspective.
	Business processes modeled with WebSphere Studio Application Developer Integration Edition Version 5.0 or earlier	Business processes modeled with WebSphere Studio Application Developer Integration Edition Version 5.0 or earlier need to be migrated to a BPEL-based process. Use the Migrate option provided with WebSphere Studio Application Developer Integration Edition Version 5.1.
	Several process choreographer API interfaces and methods used for business processes created with WebSphere Studio Application Developer Integration Edition Version 5.0 or earlier. A list can be found in the API documentation provided with process choreographer.	Information on the recommended migration action for the deprecated APIs is available in the API documentation for the appropriate API.
	JDOM (a Java representation of an XML document that provides an API for efficient reading, manipulating, and writing documentation) The currently packaged version of JDOM in WebSphere Application Server will not be packaged in subsequent releases.	Go to the JDOM website, get the latest copy of JDOM, and bundle it inside your application. **Note:** Customers running WebSphere Studio Application Developer Integration Edition Version 4.1 applications will need to migrate them to WebSphere Studio Application Developer Integration Edition Version 5.0.
	The C++ Object Request Broker (ORB), the C++ library for IDL value types and the WebSphere Application Server C++ security client Support is no longer available for the Common Object Request Broker Architecture (CORBA) C++ Developer Kit. The CORBA technology is a bridge for migration to a Java 2 Platform Enterprise Edition (J2EE) and WebSphere Application Server environment. In addition to the preceding information, the CORBA C++ client feature will be removed from the Application Clients installation image in subsequent releases.	It is recommended that customers migrate to the Object Request Broker (ORB) service for Java technology that ships with WebSphere Application Server. However, there is no equivalent J2EE functionality for the C++ security client or the C++ value-type library. Customers that require such functionality must provide or develop their own. The deprecation of the CORBA C++ Developer Kit does not affect support for CORBA interoperability with vendor software for CORBA services. View the following links for additional information about interoperability: • CORBA Interoperability Samples documentation • IBM WebSphere Application Servers CORBA Interoperability white paper
	IBM Cloudscape Version 5.1.x	Use the Cloudscape Network Server JDBC driver.
Server	IBM HTTP Server (IHS) Version 1.3.x	If you are using IHS Version 1.3.x with modules that: • are included as part of IHS Version 1.3.x packages, you do not need to take any action to migrate those modules. • are supplied by a third party (including other IBM products), you need to obtain IHS/Apache 2 versions of these modules from the third party. • have been customized or are inhouse, you need to port these modules to the new IHS/Apache 2 API.

Table 135. Features deprecated in Version 5.1 (continued). This table describes the features that are deprecated in Version 5.1.

Category	**Deprecation**	**Recommended Migration Action**
Application programming model and container support	Bean Scripting Framework (BSF) JavaServer Pages (JSP) execution and debugging functionality	The functionality will need to be rearchitected if you are using the JavaScript, Tcl, and Python languages. If using BSF scripting in your own custom applications, they will be unaffected. Custom scripts written for the WebSphere Application Server administrative console will also be unaffected. This functionality will continue to exist in WebSphere Application Server Version 5.1 and succeeding releases until Version 6.0. If debugging JSP files, you might have to restart the application server during JavaScript debugging sessions.
	The following Business Rule Bean classes, methods, and attributes: • Public classes: – com.ibm.websphere.brb.RuleImporter – com.ibm.websphere.brb.RuleExporter • Public method: – getLocalRuleManager() on class com.ibm.websphere.brb.TriggerPoint • Protected attribute: – ruleMgr on class com.ibm.websphere.brb.TriggerPoint	Users do not have to take any action.

	Data access programming interfaces in com.ibm.websphere.rsadapter.
	Relational resource adapter interface: (com.ibm.websphere.rsadapter).
	Methods are deprecated in the following types:
	`com.ibm.websphere.rsadapter.OracleDataStoreHelper`
	`public void doSpecialBLobWork(ResultSet rset,`
	`InputStream[] data, String[] blobColumnNames)`
	`public String assembleSqlString(String[] blobColumnNames,`
	`StringBuffer whereClause, String[] varValues,`
	`String tableName)`
		These relational resource adapter deprecated methods do not impact the application.
		Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of OracleDataStoreHelper class. Those deprecated methods will not be called by the WebSphere Application Server runtime.

	Scheduler (com.ibm.websphere.scheduler) programming interfaces -- Version 5.x public types in:
	• Interface methods
	`scheduler.Scheduler`
	`public BeanTaskInfo createBeanTaskInfo();`
	`public MessageTaskInfo createMessageTaskInfo();`
		Use the following methods instead of the deprecated methods:
		`public Object createTaskInfo(Class`
		`taskInfoInterface) throws TaskInfoInvalid;`
		To create a BeanTaskInfo object using the supported createTaskInfo methods:
		`BeanTaskInfo ti = (BeanTaskInfo)`
		`Scheduler.createTaskInfo(BeanTaskInfo.class);`

	Web container API modifications:
	Note: There are no declared deprecations. The only changes are caused because of a Java API that changed between 1.3 and 1.4.
	The changed class is com.ibm.websphere.servlet.error.ServletErrorReport. The return signature for getStackTrace() is changed because java.lang.Throwable now defines the same method with a different return signature.
	• Previous method signature
	`public String getStackTrace();`
	`// returns a String representation of the`
	`exception stack`
	• New method signature (Java Development Kit 1.4, WebSphere Application Server 5.1)
	`public StackTraceElement[] getStackTrace();`
	`// returns an array of stack trace`
	`elements`
	• Replacement method (5.1) (a replacement method that carries on the old functionality has been provided):
	`public String getStackTraceAsString();`
	`// returns a String representation`
	`of the Exception Stack`
		If you are using com.ibm.websphere.servlet.error.ServletErrorReport.getStackTrace() and expecting a return type of String, you need to change your application to use the replacement method.

Table 135. Features deprecated in Version 5.1 (continued). This table describes the features that are deprecated in Version 5.1.

Category	Deprecation	Recommended Migration Action
Application services	Data access binaries -- Common Connector Framework, including the following JAR files: • `ccf.jar` • `ccf2.jar` • `recjava.jar` • `eablib.jar`	The J2EE Connector Architecture solution should be used instead of the Common Connector Framework.
	Setting the XA partner log directory using the TRANLOG_ROOT variable	The setting currently stored in the TRANLOG_ROOT variable (if any) will need to be added to the Transaction Service panel for all servers that need to use the XA partner log. If the default location is to be used, then no action is required. The Transaction Service panel can be found on the administrative console by selecting Application Servers on the left, choosing the application server to be modified, and selecting Transaction Service on the panel that is displayed. The directory currently in TRANLOG_ROOT should be entered in the Logging Directory box on the panel.
Security	API for com.ibm.websphere.security.auth.WSPrincipal.getCredential().	Instead of getting the WSCredential from the principal, you should now use one of the following methods to get the Subject that contains the WSCredential: • The RunAs Subject is the Subject used for outbound requests. • The Caller subject is the Subject that represents the authenticated caller for the current request. • The methods to use to get the runAs and caller subjects are as follows: com.ibm.websphere.security.auth.WSSubject.getRunAsSubject() and com.ibm.websphere.security.auth.WSSubject.getCallerSubject() respectively.
	The following elements in the security programming interface: • The interface is deprecated in com.ibm.websphere.security.auth.WSSecurityContext. • The exception is deprecated in com.ibm.websphere.security.auth.WSSecurityContextException. • The class is deprecated in com.ibm.websphere.security.auth.WSSecurityContextResult.	Use Java Authentication and Authorization Service (JAAS) for all authentication related functionality.
	Integrated Cryptographic Services Facility (ICSF) authentication mechanism	Use the Lightweight Third-Party Authentication (LTPA) mechanism.
System administration	The following class: com.ibm.websphere.rsadapter.DB2390DataStoreHelper	If you currently use the DB2390DataStoreHelper class for the DB2 Legacy CLI-based provider when you are accessing data, you should now use the DB2DataStoreHelper class. If you currently use the DB2390DataStoreHelper class for the DB2 Universal JDBC provider when you are accessing data, you should now use the DB2UniversalDataStoreHelper class.

Features deprecated in Version 5.0.2

IBM i

Table 136. Features deprecated in Version 5.0.2. This table describes the features that are deprecated in Version 5.0.2.

Category	Deprecation	Recommended Migration Action
Application Programming Model and Container Support	Apache SOAP channel in web services gateway.	Gateway services should be deployed to the SOAP HTTP channel instead of the Apache SOAP channel. The Endpoint (URL) of the service will be different for this channel and therefore client programs that are talking to the gateway will need to use the new service Endpoint.
	Apache SOAP, WEBSJAVA.SOAP: • `soap.jar` • `wssoap.jar`	See the information center for more information.

	Scheduler (com.ibm.websphere.scheduler) programming interfaces -- Version 5.x public types in:
	• interface method
	`scheduler.MessageTaskInfo`
	`public int setJMSPriority();`
		Use the following method instead of the deprecated method:
		`public int getJMSPriority();`

Application Services	Data access programming interfaces in com.ibm.websphere.rsadapter.
	Relational resource adapter interface (com.ibm.websphere.rsadapter)
	Methods have been deprecated in these types:
	`com.ibm.websphere.rsadapter.DataStoreHelper`
	`public int processSQL(java.lang.String.sqlString,`
	`int isolevel, boolean addForUpdate,`
	`boolean addextendedForUpdateSyntax);`
	`public DataStoreAdatperException`
	`mapException(DataStoreAdapterException e);`
	`com.ibm.websphere.rsadapter.GenericDataStoreHelper`
	`public int processSQL(java.lang.String.sqlString,`
	`int isolevel, boolean addForUpdate,`
	`boolean addextendedForUpdateSyntax);`
	`public DataStoreAdatperException`
	`mapException(DataStoreAdapterException e);`
	`com.ibm.websphere.rsadapter.WSCallHelper`
	`public static DataStoreHelper`
	`createDataStoreHelper(String dsClassName)`
		These relational resource adapter deprecated methods do not impact the application.
		Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of GenericDataStoreHelper. Those deprecated methods will not be called by the WebSphere Application Server runtime.
		For com.ibm.websphere.rsadapter.WSCallHelper, use the getDataStoreHelper(datasource) method to get a DataStoreHelper object.

System Administration	DB2390DataStoreHelper and DB2390LocalDataStoreHelper classes	The DB2DataStoreHelper class now gives all the required helper information needed for the providers that currently use the DB2390DataStoreHelper and the DB2390LocalDataStoreHelper classes.
	`testConnection` command in the AdminControl scripting object ($AdminControl TestConnection configId props) Running this command in WebSphere Application Server, Version 5.0.2 or later returns the following message: `WASX7390E: Operation not supported - testConnection command with config id and properties arguments is not supported. Use testConnection command with config id argument only.`	As of WebSphere Application Server, Version 5.0.2 or later, the preferred way to test a data source connection is the testConnection command passing in the data source configuration ID as the only parameter.
	`getPropertiesForDataSource` command in the AdminControl scripting object (`$AdminControl getPropertiesForDataSource configId`) This command incorrectly assumes the availability of the configuration service when you run it in the connected mode. Running this command in WebSphere Application Server, Version 5.0.2 or later returns the following message: `WASX7389E: Operation not supported - getPropertiesForDataSource command is not supported.`	There is no replacement for this command.

Features deprecated in Version 5.0.1

Table 137. Features deprecated in Version 5.0.1. This table describes the features that are deprecated in Version 5.0.1.

Category	Deprecation	Recommended Migration Action

Application Services	Data access programming interfaces in com.ibm.websphere.rsadapter.
	Relational resource adapter interface (com.ibm.websphere.rsadapter).
	Methods have been deprecated in these types:
	`com.ibm.websphere.rsadapter.DataStoreHelper`
	`public int processSQL(java.lang.String sqlString,`
	`int isolevel);`
	`com.ibm.websphere.rsadapter.GenericDataStoreHelper`
	`public int processSQL(java.lang.String sqlString,`
	`int isolevel);`
	`com.ibm.websphere.rsadapter.DB2390DataStoreHelper`
	`public int processSQL(java.lang.String sqlString,`
	`int isolevel);`
		These relational resource adapter deprecated methods do not impact the application.
		Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of com.ibm.websphere.rsadapter.GenericDataStoreHelper. Those deprecated methods will not be called by the WebSphere Application Server runtime.

Features deprecated in Version 5.0

Table 138. Features deprecated in Version 5.0. This table describes the features that are deprecated in Version 5.0.

Category	Deprecation	Recommended Migration Action

Application Services	The following three methods from com.ibm.websphere.appprofile.accessintent.AccessIntent:
	`public boolean getPessimisticUpdateHintWeakestLockAtLoad();`
	`public boolean getPessimisticUpdateHintNoCollision();`
	`public boolean getPessimisticUpdateHintExclusive();`
	This is a base API.
		Rather than using the three deprecated methods on the AccessIntent interface, developers should use the following method from the same interface:
		`public int getPessimisticUpdateLockHint();`
		The possible return values are defined on the AccessIntent interface:
		`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;`
		`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;`
		`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;`
		`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;`

	Web application programming interfaces -- Various Version 5.x methods in com.ibm.websphere.ServletErrorReport	
Security	com.ibm.websphere.security.CustomRegistry interface	Use the com.ibm.websphere.security.UserRegistry interface.
Performance	Performance Monitoring Infrastructure -- Various Version 5.x public methods in: • com.ibm.websphere.pmi.stat.StatsUtil • com.ibm.websphere.pmi.PmiJmxTest • com.ibm.websphere.pmi.client.PmiClient	These methods are replaced as follows: • com.ibm.websphere.pmi.stat.StatsUtil There is no replacement for StatsUtil. • com.ibm.websphere.pmi.PmiJmxTest Use PmiClient.findConfig() instead. • com.ibm.websphere.pmi.client.PmiClient The getNLSValue (String key) is replaced with getNLSValue (String key, String moduleID).

Stabilized features

If you are migrating your configuration from an earlier release of WebSphere Application Server, you should be aware of the various features that have been stabilized in this release.

If a feature is listed here as stabilized, IBM does not currently plan to deprecate or remove this capability in a subsequent release of the product; but future investment will be focused on the alternative function listed under *Strategic Alternative*. You do not need to change any of your existing applications and scripts that use a stabilized function; but you should consider using the strategic alternative for new applications.

Features stabilized in Version 8.5

Table 139. Features stabilized in Version 8.5. This table describes the features that are stabilized in Version 8.5.

Category	Stabilized Function	Strategic Alternative
Application programming model and container support	Use of the Common Base Event application-programming interface (API) for problem determination • Classes or Interfaces: – All classes and interfaces from org.eclipse.hyades.logging.* packages • Methods: – Use of CommonBaseEvent or CommonBaseEventLogRecord as a message parm in the following WsJDK14Logger methods: - public void trace(Object message) - public void trace(Object message, Throwable t) - public void debug(Object message) - public void debug(Object message, Throwable t) - public void info(Object message) - public void info(Object message, Throwable t) - public void warn(Object message) - public void warn(Object message, Throwable t) - public void error(Object message) - public void error(Object message, Throwable t) - public void fatal(Object message) - public void fatal(Object message, Throwable t) • Other: – Use of the eventfactory Logger.properties logger extension - CBE-XML-1.0.1 output format for showlog - CBE-1.0.1 output format for HPEL logviewer – Use of the custom property name="com.ibm.ws.logging.zOS.errorLog.format" and value "CBE-XML-1.0.1" – Use of the showlog command on z/OS	None

Features stabilized in Version 8.0

Table 140. Features stabilized in Version 8.0. This table describes the features that are stabilized in Version 8.0.

Category	Stabilized Function	Strategic Alternative
Application programming model and container support	ActiveX to Enterprise JavaBeans (EJB) Bridge	Do not use Active X to access EJB.
	WebSphere Application Servers V4 Data sources and ConnectionManager	Use the WebSphere Application Servers Data sources (non-V4) and ConnectionManager.
	Use of CommonBaseEventLogRecord for logging	Use standard java.util.logging API for logging; and when needed, use High Performance Extensible Logging (HPEL) log and trace facility's LogViewer command to convert log and trace messages into Common Base Event XML.
System administration	WebSphere Application Server Reliability, Availability, and Serviceability (RAS) basic logging formats—`System.out`, `System.err`, `trace.log`, and `activity.log`	Use the High Performance Extensible Logging (HPEL) log and trace facility to improve logging performance as well as to improve analysis and merging of logs.

Features stabilized in Version 7.0

Table 141. Features stabilized in Version 7.0. This table describes the features that are stabilized in Version 7.0.

Category	Stabilized Function	Strategic Alternative
Application programming model and container support	Enterprise JavaBeans (EJB) entity beans: Container-Managed Persistence (CMP) 1.x and 2.x, and Bean-Managed Persistence (BMP)	Use the Java Persistence API (JPA) for new database and other persistence-related operations.
	Java API for XML-based RPC (JAX-RPC) The Java Community Process (JCP) is limiting the focus for enhancements to the JAX-RPC runtime for building web services; therefore, WebSphere Application Server will follow suit and limit enhancements.	Java API for XML Web Services (JAX-WS) will become the strategic runtime on which any new enhancements will be focused. The focus to ensure interoperability for the subset of capabilities that map to the JAX-RPC and JAX-WS intersection will be maintained; but all new enhancements related to updating to support new standards will be only in the JAX-WS runtime.
System administration	Application server administrative (wsadmin) scripting support for the Jacl language	Use Jython syntax for any new wsadmin scripting.
J2EE resources	Support for configuring and using message-driven beans (MDBs) through JMS listener ports	Perform the following actions to use JMS activation specifications instead of listener ports: • Create a JMS activation specification to replace the listener port. • Modify the configuration of the application's Message Driven Bean listener bindings to use the activation specification instead of the listener port. • Because an JMS activation specification can be defined at a wider scope than a listener-port definition (which is restricted to server scope), you might be able to replace multiple listener-port definitions with a single activation specification. • Update any administrative scripts that define or administer listener ports to define or administer JMS activation specifications instead. • Update any administrative scripts that use the stop or start operations of the ListenerPort MBean to use the pause and resume operations on the Message Endpoint MBean instead.

Removed features

If you are migrating your configuration from an earlier release of WebSphere Application Server, you should be aware of the various features that have been removed from this and earlier releases.

If a feature is listed as deprecated in “Deprecated features” on page 1105, IBM might remove this capability in a subsequent release of the product. Future investment will be focussed on the strategic function listed under *Recommended Migration Actions* in “Deprecated features” on page 1105. Typically, a feature is not removed until at least two major releases or three full years (whichever time period is longer) after the release in which that feature is deprecated. For example, features that are deprecated in Version 6.0, Version 6.0.1, or Version 6.0.2 are not removed from the product until after Version 7.0 because both Version 6.0.x and Version 6.1.x are major releases. In rare cases, it might become necessary to remove features sooner; such cases are indicated clearly and explicitly in the descriptions of these deprecated features in “Deprecated features” on page 1105.

The following tables describe what is removed—such as features, APIs, scripting interfaces, tools, wizards, publicly exposed configuration data, naming identifiers, and constants. Where possible, the recommended replacement is identified.

- “Features removed in Version 8.5” on page 1139
- “Features removed in Version 8.0” on page 1141
- “Features removed in Version 7.0” on page 1143
- “Features removed in Version 6.1” on page 1145
- “Features removed in Version 6.0” on page 1146

Features removed in Version 8.5

IBM i

Table 142. Features removed in Version 8.5. This table describes the features that are removed in Version 8.5.

Feature	Recommended Migration Action
`wasprofile` command	Use the `manageprofiles` command.
`crtjavapgm` command	No migration action is necessary.
The waslogbr and wcslogbr tools, and associated scripts to launch the tools	No migration action is necessary. Log files can be viewed using the administrative console or any text editor.
Support for the IBM WebSphere Application Server Feature Pack for Dynamic Scripting	Migrate IBM WebSphere sMash applications to the Liberty profile, where they can take advantage of the small-footprint runtime and industry-standard RESTful services using JAX-RS as well as gain access to significant updates in DOJO support.

Features removed in Version 8.0

IBM i

Table 143. Features removed in Version 8.0. This table describes the features that are removed in Version 8.0.

Feature	**Recommended Migration Action**
Support for IBM Java Developer Kit for IBM i, which is also referred to as Classic JVM	Use IBM Technology for Java on IBM i, which includes IBM Java Standard Edition (SE) 32-bit and IBM Java SE 64-bit.
Apache SOAP channel in web services gateway	Gateway services should be deployed to the SOAP HTTP channel instead of the Apache SOAP channel. The endpoint (URL) of the service will be different for this channel; and therefore ,client programs that are talking to the gateway will need to use the new service endpoint.
Apache SOAP, WEBSJAVA.SOAP • `soap.jar` • `wssoap.jar`	Migrate web services that were developed using Apache SOAP to Java API for XML-based RPC (JAX-RPC) web services that are developed based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification. See the information center for more information.
The following classes and fields of the WebSphere relational resource adapter: • Class com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper • Field com.ibm.websphere.rsadapter.DataStoreHelper.ORACLE_10G_HELPER • Class com.ibm.websphere.rsadapter.OracleDataStoreHelper • Field com.ibm.websphere.rsadapter.DataStoreHelper.ORACLE_HELPER	If you are using the Oracle10gDataStoreHelper, ORACLE_10G_HELPER, OracleDataStoreHelper, or ORACLE_HELPER, switch to the Oracle 11g JDBC driver and use the Oracle11gDataStoreHelper or ORACLE_11G_HELPER instead.

Features removed in Version 7.0

IBM i

Table 144. Features removed in Version 7.0. This table describes the features that are removed in Version 7.0.

Feature	**Recommended Migration Action**
Support for the following interfaces: • Java Virtual Machine Profiler Interface (JVMPI) • Java Virtual Machine Debug Interface (JVMDI)	Use the Java Virtual Machine Tool Interface (JVMTI). For more information, read JVM Tool Interface (JVMTI).
All classes in the com.ibm.websphere.servlet.filter package: • ChainedRequest • ChainedResponse • ChainerServlet • ServletChain	Rearchitect your applications to use javax.servlet.filter classes rather than com.ibm.websphere.servlet.filter classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. You can also chain functionality as well as embellish and truncate responses.
Integrated Cryptographic Services Facility (ICSF) authentication mechanism	Use the Lightweight Third-Party Authentication (LTPA) mechanism.
The following Java Database Connectivity (JDBC) drivers: • WebSphere Connect JDBC driver • Microsoft SQL Server 2000 Driver for JDBC • WebSphere SequeLink JDBC driver for Microsoft SQL Server	Use the DataDirect Connect JDBC driver or Microsoft SQL Server JDBC driver. Review Data source minimum required settings, by vendor for specific JDBC providers. Read the *Migrating from the WebSphere Connect JDBC driver* article in the information center for information on using the **WebSphereConnectJDBCDriverConversion** command to convert data sources from the WebSphere Connect JDBC driver to the DataDirect Connect JDBC driver or the Microsoft SQL Server JDBC driver..
Support for the DB2 legacy CLI-based Type 2 JDBC Driver and the DB2 legacy CLI-based Type 2 JDBC Driver (XA) For more information, read Support for DB2 legacy CLI-based Type 2 JDBC Drivers is removed from IBM WebSphere Application Server Version 7.0.	Use the DB2 Universal JDBC Driver.
mb2mdb command-line utility	No migration action is necessary.
Web services gateway customization API	Replace your existing filters with a combination of JAX-RPC handlers and service integration bus mediations.
com.ibm.websphere.servlet.session.UserTransactionWrapper class	Store a UserTransaction directly into the HTTP session without wrapping it in the removed class.
com.ibm.websphere.rsadapter.DataDirectDataStoreHelper class	Use the com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper class.
com.ibm.websphere.rsadapter.MSSQLServerDataStoreHelper class	Use the com.ibm.websphere.rsadapter.MicrosoftSQLServerDataStoreHelper class.
Derby Network Server Provider using the Universal JDBC driver	Use the Derby Network Server using Derby Client instead.
Support for the following custom properties: • com.ibm.security.SAF.unauthenticatedId • com.ibm.security.SAF.useEJBROLEAuthz • com.ibm.security.SAF.useEJBROLEDelegation	Use the following custom properties that are specified on the SAF authorization options panel: • com.ibm.security.SAF.unautheticated • com.ibm.security.SAF.authorization • com.ibm.security.SAF.delegation

Features removed in Version 6.1

IBM i

Table 145. Features removed in Version 6.1. This table describes the features that are removed in Version 6.1.

Feature	Recommended Migration Action
com.ibm.websphere.security.CustomRegistry interface	Use the com.ibm.websphere.security.UserRegistry interface.
Support for the Secure Authentication Service (SAS) IIOP security protocol	Use the Common Secure Interoperability Version 2 (CSIv2) protocols.
Support for the Secure Authentication Service (SAS) CORBA security programming APIs	Migrate from the SAS programming APIs to the Java Authentication and Authorization Service (JAAS).
Support for the Common Connector Framework (CCF)	Use the J2EE Connector Architecture (JCA) solution.
Support for the IBM Cloudscape Version 5.1.x database	Use the IBM Cloudscape Version 10.1 database. This database provides Derby Version 10.1 binaries, National Language Support (NLS) enablement, quality assurance, and product support. The term "Derby" rather than "Cloudscape" is used in places such as the administrative console, the **`ejbdeploy`** command, and others.
Log Analyzer, the tool that was previously provided for viewing and analyzing the activity or service log file	Use the Log and Trace Analyzer tool for Eclipse in the Application Server Toolkit. This tool is installable from the Application Server Toolkit launchpad console.
Mozilla Rhino JavaScript (`js.jar` file)	Use the Rhino code available from Mozilla. Go to the Rhino: JavaScript for Java website for the latest copy of Rhino.
Java Document Object Model (JDOM)	Use the code available from the JDOM organization. Go to the JDOM website, download the latest copy of JDOM, and bundle it inside your application.
Class preloading function	No migration action is necessary.
The following samples from the Samples Gallery: • Adventure Builder • Greenhouse by WebSphere • WebSphere Bank The following technology samples from the Samples Gallery: • Bean-Managed Persistence (BMP) • Container-Managed Persistence (CMP) 1.1 • Container-Managed Persistence (CMP) 2.1 • Container-Managed Relationships (CMR) • EJB Time • Filter Servlet • JavaServer Pages (JSP) 2.0 • Message-Driven Beans (MDB) • Pagelist Servlet • Simple JavaServer Pages (JSP) • Simple Servlet • Stateful Session • TagLib	No migration action is necessary.

Features removed in Version 6.0

Table 146. Features removed in Version 6.0. This table describes the features that are removed in Version 6.0.

Component	Classes and Interfaces
Activity	com.ibm.ws.activity.ActivityConstants com.ibm.ws.activity.ActivityService com.ibm.ws.activity.ActivityServiceInitializer com.ibm.ws.activity.ActivityTrace com.ibm.ws.activity.GlobalIdImpl com.ibm.ws.activity.HighlyAvailableServiceManager com.ibm.ws.activity.HLSLiteDataInterface com.ibm.ws.activity.HLSLiteExtended com.ibm.ws.activity.HLSLiteInfo com.ibm.ws.activity.j2ee_activity_specific_data com.ibm.ws.activity.j2ee_activity_specific_dataHelper com.ibm.ws.activity.ServiceMigration com.ibm.ws.activity.VUTrace com.ibm.ws.activity.WebSphereServiceManager com.ibm.ws.activity.WebSphereUserActivity com.ibm.ws.javax.activity.ActionErrorException com.ibm.ws.javax.activity.ActionNotFoundException com.ibm.ws.javax.activity.ActivityCoordinator com.ibm.ws.javax.activity.ActivityInformation com.ibm.ws.javax.activity.ActivityManager com.ibm.ws.javax.activity.ActivityNotProcessedException com.ibm.ws.javax.activity.ActivityPendingException com.ibm.ws.javax.activity.ActivityToken com.ibm.ws.javax.activity.CompletionStatus com.ibm.ws.javax.activity.ContextPendingException com.ibm.ws.javax.activity.CoordinationInformation com.ibm.ws.javax.activity.GlobalId com.ibm.ws.javax.activity.InvalidParentContextException com.ibm.ws.javax.activity.InvalidStateException com.ibm.ws.javax.activity.NoActivityException com.ibm.ws.javax.activity.NoImplementException com.ibm.ws.javax.activity.NotOriginatorException com.ibm.ws.javax.activity.Outcome com.ibm.ws.javax.activity.PersistentActivityCoordinator com.ibm.ws.javax.activity.PropertyGroupContext com.ibm.ws.javax.activity.PropertyGroupRegisteredException com.ibm.ws.javax.activity.PropertyGroupUnknownException com.ibm.ws.javax.activity.ServiceAlreadyRegisteredException com.ibm.ws.javax.activity.ServiceInformation com.ibm.ws.javax.activity.ServiceNotRegisteredException com.ibm.ws.javax.activity.Signal com.ibm.ws.javax.activity.SignalSetActiveException com.ibm.ws.javax.activity.SignalSetInactiveException com.ibm.ws.javax.activity.SignalSetUnknownException com.ibm.ws.javax.activity.Status com.ibm.ws.javax.activity.SystemException com.ibm.ws.javax.activity.TimeoutRangeException com.ibm.ws.javax.activity.UserActivity com.ibm.ws.javax.activity.coordination.Action com.ibm.ws.javax.activity.coordination.RecoverableAction com.ibm.ws.javax.activity.coordination.ServiceManager com.ibm.ws.javax.activity.coordination.SignalSet com.ibm.ws.javax.activity.coordination.SubordinateSignalSet com.ibm.ws.javax.activity.propertygroup.PropertyGroup com.ibm.ws.javax.activity.propertygroup.PropertyGroupManager com.ibm.ws.javax.ejb.ActivityCompletedLocalException com.ibm.ws.javax.ejb.ActivityRequiredLocalException com.ibm.ws.javax.ejb.InvalidActivityLocalException
ALS	com.ibm.websphere.als.BufferManager
Ant tasks	com.ibm.websphere.ant.tasks.endptEnabler.Property com.ibm.websphere.ant.tasks.Java2WSDL.Mapping com.ibm.websphere.ant.tasks.Messages com.ibm.websphere.ant.tasks.WSDL2Java.Mapping
Asynchronous Beans APIs	com/ibm/websphere/asynchbeans/pmi/AlarmManagerPerf.java com/ibm/websphere/asynchbeans/pmi/AsynchBeanPerf.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorManagerPerf.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorPerf.java com/ibm/websphere/asynchbeans/pmi/AlarmManagerPmiModule.java com/ibm/websphere/asynchbeans/pmi/AsynchBeanPmiModule.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorManagerPmiModule.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorPmiModule.java

Table 146. Features removed in Version 6.0 (continued). This table describes the features that are removed in Version 6.0.

Component	Classes and Interfaces
Dynacache	com.ibm.websphere.servlet.cache.CacheConfig
Management	com.ibm.websphere.management.application.EarUtils
ObjectPool APIs	com/ibm/websphere/objectpool/pmi/ObjectPoolPerf.java com/ibm/websphere/objectpool/pmi/ObjectPoolPmiModule.java
RAS	com.ibm.ras.RASConsoleHandler com.ibm.ras.RASEnhancedMessageFormatter com.ibm.ras.RASEnhancedTraceFormatter com.ibm.ras.RASErrorHandler com.ibm.ras.RASFileHandler com.ibm.ras.RASFormatter com.ibm.ras.RASHandler com.ibm.ras.RASMessageFormatter com.ibm.ras.RASMultiFileHandler com.ibm.ras.RASSerialFileHandler com.ibm.ras.RASSocketHandler com.ibm.ras.RASTextAreaHandler com.ibm.ras.RASTraceFormatter com.ibm.websphere.ras.WsOrbRasManager
Scheduler API	com.ibm.websphere.scheduler.pmi.SchedulerPmiModule com.ibm.websphere.scheduler.pmi.SchedulerPerf com.ibm.websphere.scheduler.MessageTaskInfo.setJMSPriority()
Security	com.ibm.websphere.security.AuthorizationTable com.ibm.websphere.security.FileRegistrySample com.ibm.websphere.security.SecurityProviderException com.ibm.websphere.security.WASPrincipal com.ibm.websphere.security.auth.AuthDataFileEnc
Userprofile	com.ibm.websphere.userprofile.UserProfile com.ibm.websphere.userprofile.UserProfileCreateException com.ibm.websphere.userprofile.UserProfileExtender com.ibm.websphere.userprofile.UserProfileFinderException com.ibm.websphere.userprofile.UserProfileManager com.ibm.websphere.userprofile.UserProfileProperties com.ibm.websphere.userprofile.UserProfileRemoveException

Chapter 47. Development and assembly tools

You can use an Integrated Development Environment to develop, assemble, and deploy Java Platform, Enterprise Edition (Java EE) modules for WebSphere Application Server.

The IBM Rational Application Developer for WebSphere Software product and the IBM WebSphere Application Server Developer Tools for Eclipse product are supported tools for integrated development environments.

This information center refers to the products as the *assembly tools*. However, you can use the products to do more than assemble modules. Use these tools in an integrated development environment to develop, assemble, and deploy Java EE modules.

The Rational Application Developer for WebSphere Software is a more extensive set of tools supporting enterprise development. This workbench has integrated support for WebSphere Application Server Version 6.1 and later. This workbench also supports both the OSGi and Java EE programming models, and contains wizards and visual editors to help you develop Web 2.0, Service Component Architecture (SCA), Java, and Java EE applications. This product contains code quality tools to help you analyze code and improve performance. This product integrates with Rational Team Concert to provide a team-based environment to help developers share information and work collaboratively. The Trial download for Rational Application Developer is available at `http://www.ibm.com/developerworks/downloads/r/rad/`.

IBM WebSphere Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java EE applications to WebSphere Application Server Version 7.0 and 8.x. This workbench integrates with the application server to help you to quickly deploy and test applications. This product contains wizards and visual editors that support the Java EE programming model.

For documentation on the tools, see “Rational Application Developer documentation.” Topics on application assembly in this information center supplement that documentation.

Important: The assembly tools run on Windows and Linux Intel platforms. Users of WebSphere Application Server on all platforms must assemble their modules using an assembly tool installed on Windows or Linux Intel platforms. To install an assembly tool, follow instructions available with the tool.

Chapter 48. Web resources for learning

This topic familiarizes you with the many websites containing technical information for understanding and using your WebSphere Application Server product. A wealth of online information is available to complement the product documentation.

Choose an area of interest.

- Learning and education
- Developer resources
- Architect, planner, installer, and administrator resources
- Partner resources
- Redbooks, white papers, and documentation
- Troubleshooting and support

Also, throughout the documentation, you will find additional resources for learning pages, each focused on a specific technology, such as web services. The pages provide links to particular documents of interest.

Learning and education

IBM Education Assistant

Find tutorials, multimedia demonstrations, and presentations for WebSphere servers and Rational development tools.

Training and certification

Use this page to find educational opportunities to learn about WebSphere software. IBM has several educational options available to you. From classroom courses to on-site assistance and Internet-based training, if you are ready to learn, we are ready to teach.

Developer resources

developerWorks - WebSphere Application Server zone

Use this page to search for information, download software including trial code and fixes, learn about the application server, and find support and migration information.

Samples

The Samples section of the information center offers a set of samples that demonstrate common web application tasks.

WASdev community downloads page

Use this page to participate in the WASdev community for developers. Interact with the WebSphere and Rational Development teams, collaborate with other developers, and provide requirements input directly to the development and product management teams.

Architect, planner, installer, and administrator resources

Detailed system requirements page

These pages describe the minimum product levels you should have installed before opening a problem report with the WebSphere Application Server support team.

Patterns for e-business

Patterns for e-business are a group of reusable assets that can help speed the process of developing Web-based applications. The Patterns leverage the experience of IBM architects to create solutions quickly, whether for a small local business or a large multinational enterprise.

Partner resources

PartnerWorld®

Find product, business, and technical information. The PartnerWorld program is designed to offer IBM Business Partners benefits, technical support, education, marketing campaigns, sales tools and more to help you grow your business and drive profits.

Redbooks, white papers, and documentation

Redbooks - WebSphere

Find Redbooks pertaining to WebSphere, including the newest, latest, and most popular Redbooks and Redpapers in draft and published form.

White papers

This link performs a query for white papers that are relevant to WebSphere Application Server.

Library page

A new, improved web page for finding product documentation, including the online information center, documentation plug-ins for offline viewing with the WebSphere help system, and PDF books. This page links to a variety of other kinds of product information, such as WebSphere Redbooks.

Troubleshooting and support

IBM Support Assistant

Looking for ways to simplify software support, reduce support costs and improve your ability to resolve software problems inhouse quickly? If so, we invite you to explore IBM Support Assistant.

IBM Support Assistant allows you to search multiple knowledge repositories and gives you access to the latest product information. You can choose to be guided through your problem symptoms or view a complete listing of advanced tooling for analyzing everything from logs to memory dumps. Using the IBM Support Assistant Workbench installed on a local workstation running the Windows or Linux Intel operating system, you can connect to the IBM Support Assistant Agent installed on a remote system running on the AIX, Linux, Windows, or Solaris operating system. You can use IBM Support Assistant to run automated, symptom-specific data collectors. This data can then be attached to an IBM Service Request so that you can get help from IBM Support.

WebSphere Application Server - Support

This page provides a convenient starting point for querying technical documents, solving problems, downloading fixes, planning, learning, and communicating.

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

- IBM i WebSphere Application Server support

Support - Recent updates

This document lists valuable resources and newly created content.

Support - Resource reference list

This document is an introduction to available documentation and educational resources.

Support - Quick links

This document provides a reference of direct links to available documentation and educational resources.

Support - Recommended fixes

This document provides a comprehensive list of recommended, generally available (GA) fixes for IBM WebSphere Application Server releases.

Support - MustGather documents

MustGather documents aid in problem determination and save time resolving Problem Management Records (PMRs). Collecting MustGather data early, even before opening the PMR, helps IBM Support quickly determine if:

- Symptoms match known problems (rediscovery).
- There is a non-defect problem that can be identified and resolved.
- There is a defect that identifies a workaround to reduce severity.
- Locating root cause can speed development of a code fix.

Notes®

- The WebSphere Application Server product documentation found in the information center and PDF books documents supported configurations. Many of the above sites could contain information that describes unsupported configurations.
- Information residing on non-IBM sites is provided for your convenience. Its technical accuracy is controlled by the owner of the site. Use the information at your own risk.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

> IBM Director of Intellectual Property & Licensing
> IBM Corporation
> North Castle Drive
> Armonk, NY 10504-1785
> USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index