IBM WebSphere Application Server for z/OS, Version 8.0

Migrating WebSphere applications

GA32-0899-00

Note

Before using this information, be sure to read the general information under “Notices” on page 113.

Compilation date: June 10, 2011

© Copyright IBM Corporation 2011.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments v

Changes to serve you more quickly vii

Chapter 1. Migrating Application profiling 1
Running previous versions of application profiles on Version 8 1
Application profiling interoperability 2

Chapter 2. Migrating Asynchronous beans 3
Migrating asynchronous beans 3
Interoperating with asynchronous beans 3

Chapter 3. Migrating applications that use the Bean Validation API 5
Migration of JPA applications and bean validation 5

Chapter 4. Migrating Data access resources 7
Migrating data sources 7
Migrating applications to use data sources of the current Java EE Connector Architecture (JCA) . . . 7
Verifying Cloudscape automatic migration 11
Upgrading Cloudscape manually 14
Directory conventions 16

Chapter 5. Migrating Dynamic caching 19
Migrating servers from multi-broker replication domains to data replication domains 19

Chapter 6. Migrating EJB applications 21
Migrating enterprise bean code from Version 1.1 to Version 2.1 21
Migrating enterprise bean code to the supported specification 22
Adjusting exception handling for EJB wrapped applications migrating from version 5 to version 7 22

Chapter 7. Migrating Messaging resources 25
Migrating from WebSphere Application Server Version 5 embedded messaging 25
Migration from Version 5 embedded messaging 26
Migrating Version 5.1 messages using the message migration utility 30
Migrating a stand-alone application server from Version 5 embedded messaging 36
Migrating a network deployment configuration from Version 5 embedded messaging 39
Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 1 42
Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2 47
Using a Version 5.1 JMS client 50

Chapter 8. Migrating Naming and directory 53
Migrating bootstrap addresses 53

Chapter 10. Migrating Scheduler service 57
Interoperating with schedulers 57
Interoperating with schedulers 57

Chapter 11. Migrating Service integration 59
Preserving a Version 5.1 gateway when migrating a cell 59
Migrating a Version 5.1 Web services gateway configuration 60
Adding unique names to the bus authorization policy 63
Migrating a messaging engine based on a data store 64

© Copyright IBM Corp. 2011

Chapter 12. Migrating Transactions . . . 65
Interoperating transactionally between application servers . . . 65

Chapter 13. Migrating web applications . . . 67
Migrating web application components . . . 67
Migrating web application components from WebSphere Application Server Version 5.x . . . 67
JavaServer Faces migration . . . 71
Migration scenario for the getHeaderNames method . . . 72
JavaServer Pages migration . . . 72
JavaServer Pages migration . . . 74
JavaServer Pages migration best practices and considerations . . . 74
HTTP session migration . . . 75

Chapter 14. Migrating web services . . . 77
Migrating web services . . . 77
Web services migration scenarios: JAX-RPC to JAX-WS and JAXB . . . 77
Web services migration best practices . . . 86
Migrating Apache SOAP web services to JAX-RPC web services based on Java EE standards . . . 88
Migrating Web Services Security . . . 90
Migration of JAX-WS Web Services Security bindings from Version 6.1 . . . 90
Migrating JAX-RPC Web Services Security applications to Version 8.0 applications . . . 91
Migrating the UDDI registry . . . 107
Setting up a UDDI migration data source . . . 108
Migrating a UDDI database that uses Apache Derby . . . 109

Appendix. Directory conventions . . . 111

Notices . . . 113

Trademarks and service marks . . . 115

Index . . . 117

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 e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  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.

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

© Copyright IBM Corp. 2011

Changes to serve you more quickly

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.

Under construction!

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

- 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

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

© Copyright IBM Corp. 2011

Chapter 1. Migrating 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.

Running previous versions of application profiles on Version 8

Java 2 platform, Enterprise Edition (J2EE) 1.3 applications created using WebSphere® Application Server Version 5.x have an application profile configuration formatted for Version 5.x. Although, you can use applications with an application profile configuration from Version 5.x on WebSphere Application Server Version 7.0 and Version 8.0, you must change a setting. Also, there are several implications to using Version 5.x application profiles on Version 7.0 and Version 8.0.

About this task

The product application profiling function works under the *unit of work* concept. This gives it a more predictable data access pattern based on the active unit of work, which could be either a transaction or an ActivitySession. See the topic, Task overview: Application profiling for more information.

Procedure

- Set the Application Profile service on your server to enable the Application Profiling 5.x Compatibility Mode as the default.

  See the topic, Using the TaskNameManager interface for more information.

  Note: This setting is necessary to support Java 2 platform, Enterprise Edition (J2EE) 1.3 applications with an application profile configuration from WebSphere Application Server Version 5.x. The 5.x compatibility mode has a fair amount of performance overhead on a Version 7.0 and Version 8.0 server. Because of this, if there is no J2EE 1.3 application with an application profile V5.x configuration installed, the server *does not load* the support for the 5.x compatibility mode during startup, even when the 5.x compatibility mode is turned on.

  After the server starts without loading the 5.x compatibility mode support, if a J2EE 1.3 application with an application profile V5.x configuration installs on the server and attempts to start, the following message is displayed, and the server must be restarted:

  ```
  ACIN0031E: The J2EE 1.3 application <ApplicationName> is configured for application
  profiling and is installed and starting on a running server that enables Application Profiling
   5.x Compatibility Mode. You must re-start the server.
  ```

  This situation only happens when:
  1. the server started with the Application Profile service enabled and 5.x compatibility mode turned on, but no J2EE 1.3 applications were installed at server start up. Therefore, the server run time automatically ignores the 5.x compatibility in order to avoid performance costs associated with it.
  2. you try to install and start a J2EE 1.3 application with an application profile configured in Version 5.x, but the 5.x compatibility mode is turned off.

  To avoid this situation, you must install at least one J2EE 1.3 application with an application profile Version 5.x configuration *before* starting the server, or restart the server after installing a J2EE 1.3 application with the application profile configured in Version 5.x.
- Ideally, upgrade your J2EE 1.3 applications to use the Version 6.x application profiling configuration and turn off the Application Profiling 5.x Compatibility Mode through the administrative console.

  See the topic, Application profiling service settings, for more information. .

© IBM Corporation 2009

- Migrate any application you have configured with application profiling in Version 5.

  Application profiles migration requires you to re-configure your applications in the assembly tool. See the *Developing and deploying applications* PDF for more information.
- Rework the usage of the TaskNameManager API if it is used in your applications. The TaskNameManager API is not supported in container-managed transaction beans, and the setTaskName method must be called before beginning a new unit of work.

  See the topic, Using the TaskName Manager interface, for more information.

Application profiling interoperability

Using application profiling with 5.x compatibility mode or in a clustered environment with mixed product versions and mixed platforms can affect its behavior in different ways.

The effect of 5.x Compatibility Mode

Application profiling supports *forward* compatibility. Application profiles created in previous versions of WebSphere Application Server (Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1) can only run in WebSphere Application Server Version 6 or later if the Application Profiling 5.x Compatibility Mode attribute is turned on. If the 5.x Compatibility Mode attribute is off, Version 5 application profiles might display unexpected behavior.

For information about the 5.x Compatibility Mode, see the topic, Application profile service settings..

Similarly, application profiles that you create using the latest version of WebSphere Application Server are not compatible with Version 5 or earlier versions. Even applications configured with application profiles run on Version 6.x servers with the Application Profiling 5.x Compatibility Mode attribute turned on cannot interact with applications configured with profiles run on Version 5 servers.

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.

Considerations for a clustered environment

In a clustered environment with mixed product versions and mixed platforms, applications configured with application profiles might exhibit unexpected behavior because previous versions of server members cannot support the application profiling of Version 6.x.

If a clustered environment contains both Version 5.x and 6.x server members, and if any applications are configured with application profiles, the Application Profiling 5.x Compatibility Mode attribute must be turned on in Version 6 server members. Still, this cluster can only support Version 5 application profiling behavior. To support applications configured with Version 6 application profiles in a cluster environment, all server members in the cluster must be at the Version 6.x level.

WebSphere Application Server Enterprise Edition Version 5.0.2

If you use WebSphere Application Server Enterprise Edition 5.0.2, you must apply WebSphere Application Server Version 5 service pack 7 or later service pack to enable Application Profiling interoperability.

Chapter 2. Migrating 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.

Migrating asynchronous beans

Interoperating with asynchronous beans

Asynchronous beans support Serialized WorkWithExecutionContext interoperability with objects that are serialized in 5.0.2 or later.

Before you begin

For more information on migrating to WebSphere Application Server Version 8 from previous product releases, read the Migrating product configurations topic.

Procedure

1. Install the Version 8 product.
2. Migrate the previous release to the Version 8 product.
   Read the Overview of migration, coexistence, and interoperability topic.

© IBM Corporation 2009

Chapter 3. Migrating applications that use the Bean Validation API

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, 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.

Migration of JPA applications and bean validation

The Java Persistence API (JPA) 2.0 provides support for the Bean Validation API so that data validation can be done at run time. This topic includes best practices when you want to deploy Version 7 Feature Pack for OSGi and JPA 2.0 applications on Version 8.

The Bean Validation API was not included with the Version 7 Feature Pack for OSGi and JPA 2.0. Therefore, it was required to provide and configure the Bean Validation API and implementation in a shared library or package with the application.

Note: The Bean Validation API and implementation are included in the Version 8.0 product. If you want to deploy the feature pack applications on Version 8, 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 8.0, 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 V7 and V8 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.

© Copyright IBM Corp. 2011

Chapter 4. Migrating 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.

Migrating data sources

Migrating applications to use data sources of the current Java EE Connector Architecture (JCA)

Migrate your applications that use Version 4 data sources, or data sources (WebSphere Application Server V4), to use data sources that support more advanced connection management features, such as connection sharing.

About this task

To use the connection management infrastructure in the application server, you must package your application as a Java EE 1.3 or later application. This process involves repackaging your web modules to the 2.3 specification and your EJB modules to the 2.1 specification before installing them onto WebSphere Application Server.

Applications left at the Java EE 1.2 level will continue to run fine using the connection management support that was available at V4.0; simply create the JDBC provider and data source, and install the 4.0 application as-is. If you choose to repackage your application for version 6.0 or above you can not use a Version 4.0 data source. You must use the other data source option, which supports applications coded to the Java EE 1.3 specifications, at minimum.

Procedure

- Convert a 2.2 web module to a 2.3 web module
  1. Open an assembly tool.
  2. Create a new web module by selecting **File > New > Web Module**.
  3. Add any required class files to the new module.
     a. Expand the **Files** portion of the tree.

© Copyright IBM Corp. 2011

b. Right-click **Class Files** and select **Add Files**.
c. In the Add Files window, click **Browse**.
d. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
e. In the upper left pane of the Add Files window, navigate to your WAR file and expand the `WEB-INF` and `classes` directories.
f. Select each of the directories and files in the classes directory and click **Add**.
g. After you add all of the required class files, click **OK**.

4. Add any required JAR files to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right-click **Jar Files** and select **Add Files**.
   c. Navigate to your WebSphere 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your WAR file and expand the `WEB-INF` and `lib` directories.
   e. Select each JAR file and click **Add**.
   f. After you add all of the required JAR files, click**OK**.
5. Add any required resource files, such as HTML files, JSP files, GIFs, and so on, to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right-click **Resource Files** and select **Add Files**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your WAR file.
   e. Select each of the directories and files in the WAR file, excluding `META-INF` and `WEB-INF`, and click **Add**.
   f. After you add all of the required resource files, click**OK**.
6. Import your web components.
   a. Right-click **Web Components** and select **Import**.
   b. In the Import Components window click **Browse**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Open**.
   d. In the left top pane of the **Import Components** window, highlight the WAR file that you are migrating.
   e. Highlight each of the components that display in the right top pane and click **Add**.
   f. When all of your web components display in the Selected Components pane of the window, click **OK**.
   g. Verify that your web components are correctly imported under the Web Components section of your new web module.
7. Add servlet mappings for each of your web components.
   a. Right-click **Servlet Mappings** and select **New**.
   b. Identify a URL pattern for the web component.
   c. Select the web component from the Servlet drop-down box.
   d. Click **OK**.
8. Add any necessary resource references by following the instructions in the Creating a resource reference article in the information center.
9. Add any other web module properties that are required. Click **Help** for a description of the settings.
10. **Save** the web module.

- Convert a 1.1 EJB module to a 2.1 EJB module (or later)
  1. Open an assembly tool.

2. Create a new EJB Module by selecting **File > New > EJB Module**.
3. Add any required class files to the new module.
   a. Right-click **Files object** and select **Add Files**.
   b. In the Add Files window click **Browse**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your enterprise bean JAR file.
   e. Select each of the directories and class files and click **Add**.
   f. After you add all of the required class files, click **OK**
4. Create your session beans and entity beans. To find help on this subject, see the information center article Migrating enterprise bean code to the supported specification.
5. Add any necessary resource references by following the instructions in the Creating a resource reference article in the information center.
6. Add any other EJB module properties that are required. Click **Help** for a description of the settings.
7. **Save** the EJB module.
8. Generate the deployed code for the EJB module by clicking **File > Generate Code for Deployment**.
9. Fill in the appropriate fields and click **Generate Now**.

- Add the EJB modules and web modules to an EAR file
  1. Open an assembly tool.
  2. Create a new Application by selecting **File > New > Application**.
  3. Add each of your EJB modules.
     a. Right-click **EJB Modules** and select **Import**.
     b. Navigate to your converted EJB module and click **Open**.
     c. Click **OK**.
  4. Add each of your web modules.
     a. Right-click **Web Modules** and select **Import**.
     b. Navigate to your converted web module and click **Open**.
     c. Fill in a **Context root** and click **OK**.
  5. Identify any other application properties. Click **Help** for a description of the settings.
  6. Save the EAR file.
- Install the application on the application server.
  1. Install the application following the instructions in the topic on installing a new application, and bind the resource references to the data source that you created.
  2. Perform the necessary administrative task of creating a JDBC provider and a data source object following the instructions in the topic on creating a JDBC provider and data source.

Connection considerations when migrating servlets, JavaServer Pages, or enterprise session beans

If you plan to upgrade to WebSphere Application Server Version 6.x, and migrate applications from version 1.2 of the Java 2 Platform, Enterprise Edition (J2EE) specification to a later version, such as 1.4 or Java Platform, Enterprise Edition (Java EE), be aware that the product allocates shareable and unshareable connections differently for post-version 1.2 application components. For some applications, that difference can result in performance degradation.

Adverse behavior changes

Because WebSphere Application Server provides backward compatibility with application modules coded to the J2EE 1.2 specification, you can continue to use Version 4 style data sources when you migrate to

Application Server Version 6.x. As long as you configure Version 4 data sources *only* for J2EE 1.2 modules, the behavior of your data access application components does not change.

If you are adopting a later version of the J2EE specification along with your migration to Application Server Version 6.x, however, the behavior of your data access components can change. Specifically, this risk applies to applications that include servlets, JavaServer Page (JSP) files, or enterprise session beans that run inside local transactions over shareable connections. A behavior change in the data access components can adversely affect the use of connections in such applications.

This change affects all applications that contain the following methods:
- `RequestDispatcher.include()`
- `RequestDispatcher.forward()`
- JSP includes (`<jsp:include>`)

Symptoms of the problem include:
- Session hang
- Session timeout
- Running out of connections

Note: You can also experience these symptoms with applications that contain the components and methods described previously if you are upgrading from J2EE 1.2 modules *within* Application Server Version 6.x.

The switch in allocating shareable and unshareable connections

For J2EE 1.2 modules using Version 4 data sources, WebSphere Application Server issues non-sharable connections to JSP files, servlets, and enterprise session beans. All of the other application components are issued shareable connections. However, for J2EE 1.3 and later modules, Application Server issues shareable connections to *all* logically named resources (resources bound to individual references) unless you specify the connections as unshareable in the individual resource-references. Using shareable connections in this context has the following effects:
- All connections that are received and used outside the scope of a user transaction are *not* returned to the free connection pool until the encapsulating method returns, even when the connection handle issues a `close()` call.
- All connections that are received and used outside the scope of a user transaction are *not* shared with other component instances (that is, other servlets, JSP files, or enterprise beans). For example, session bean 1 gets a connection and then calls session bean 2 that also gets a connection. Even if all properties are identical, each session bean receives its own connection.

If you do not anticipate this change in the connection behavior, the way you structure your application code can lead to excessive connection use, particularly in the cases of JSP includes, session beans that run inside local transactions over shareable connections, `RequestDispatcher.include()` routines, `RequestDispatcher.forward()` routines, or calls from these methods to other components. Consequently, you can experience session hang, session timeout, or connection deficiency.

Example scenario

Servlet A gets a connection, completes the work, commits the connection, and calls **`close()`** on the connection. Next, servlet A calls the `RequestDispatcher.include()` to include servlet B, which performs the same steps as servlet A. Because the servlet A connection does not return to the free pool until it returns from the current method, two connections are now busy. In this way, more connections might be in use than you intended in your application. If these connections are not accounted for in the **Max Connections** setting on the connection pool, this behavior might cause a lack of connections in the pool, which results in `ConnectionWaitTimeOut` exceptions. If the **connection wait timeout** is not enabled, or if the **connection wait timeout** is set to a large number, these threads might appear to hang because they are waiting for connections that are never returned to the pool. Threads waiting for new connections do not return the

ones they are currently using if new connections are not available.

Resolution

To resolve these problems:
1. Use unshared connections.

   If you use an unshared connection and are not in a user transaction, the connection is returned to the free pool when you issue a `close()` call (assuming you commit or roll back the connection).
2. Increase the maximum number of connections.

   To calculate the number of required connections, multiply the number of configured threads by the deepest level of component call nesting (for those calls that use connections). See the Examples section for a description of call nesting.

Verifying Cloudscape automatic migration

Version 7.0 of the application server requires Cloudscape or Apache Derby to run at a minimal version of v10.1.x. During the application server upgrade to version 7.0, the migration tool automatically upgrades the database instances that are accessed through the embedded framework by some internal components, such as the UDDI registry. The tool also attempts to upgrade Cloudscape or Derby instances that your applications access through the embedded framework. You must verify the migration results for these backend databases.

Before you begin

Do not use Apache Derby or Cloudscape as a production database. Use it for development and test purposes only.

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.

The migration tool attempts to upgrade Cloudscape database instances that are accessed through the embedded framework only. You must manually upgrade Cloudscape instances that transact with application servers on the Network Server framework. See the topic, Upgrading Cloudscape manually. This requirement eliminates the risk of corrupting third party applications that use the Network Server framework to access the same database instances as WebSphere Application Server.

Other applications can access Apache Derby or Cloudscape on Network Server because the framework provides the database with a foundation of connectivity software; the embedded framework does not. Derby Network Server or Cloudscape Network Server can transact with multiple Java Virtual Machines (JVM) or application servers concurrently, whereas Cloudscape or Derby on the embedded framework works with only a single JVM. Clustered or coexistence implementations of Application Server require Network Server. For more information, consult the IBM® Cloudscape Information Center. Find the link in the following IBM Suggests section.

About this task

For database instances that your applications access through the embedded framework, the automatic migration can succeed completely, fail completely, or succeed with warnings. A migration that produces warning messages does create an Apache Derby or Cloudscape database with your data, but does not migrate all of your configured logic and other settings, such as:

- keys
- checks
- views
- triggers

- aliases
- stored procedures

To distinguish between a partially and a completely successful migration, you must verify the auto-migration results by checking both the general post-upgrade log and the individual database logs. Performing these tasks gives you vital diagnostic data to troubleshoot the partially migrated databases as well as those that fail auto-migration completely. Ultimately, you migrate these databases through a manual process.

Procedure

1. Open the post-upgrade log of each new profile for the application server. The path name of the log is *app_server_root*/profiles/*profileName*/logs/WASPostUpgrade.*timestamp*.log.
2. Examine the post-upgrade log for database error messages. These exceptions indicate database migration failures. The following lines are an example of post-upgrade log content, in which the database error code is `DSRA7600E`. The migration tool references all database exceptions with the prefix `DSRA`.

   ```
   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/cloudscape
   /db2j.properties.

   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/config/cells
   /migr06/applications/MyBankApp.ear/deployments/MyBankApp/deployment.xml.

   DSRA7600E: Cloudscape migration of database instance /opt/WebSphere61/Express
   /profiles/default/databases/_opt_WebSphere51_AppServer_bin_DefaultDB failed,
   reason: java.sql.SQLException: Failure creating target db

   MIGR0430W: Cloudscape Database /fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB failed to migrate <new database name>
   ```

   Important: Call IBM WebSphere Application Server Support if you see a migration failure message for a Cloudscape instance that is accessed by a WebSphere internal component (that is, a component that helps comprise WebSphere Application Server rather than one of your applications).
3. Open the individual database migration log that corresponds with each of your backend Cloudscape databases. These logs have the same timestamp as that of the general post-upgrade log. The logs display more detail about errors that are listed in the general post-upgrade log, as well as expose errors that are not documented by the general log.

   The path name of each database log is *app_server_root*/profiles/*profileName*/logs/*myFulldbPathName*_migrationLog*timestamp*.log.
4. Examine each database migration log for errors. For a completely successful migration, the log displays a message that is similar to the following text:

   ```
   MIGR0429I: Cloudscape Database F:\temp\51BaseXExpress\PostUpgrade50BaseFVTTest2\testRun
   \pre\websphere_backup\bin\DefaultDB was successfully migrated. See log C:\WebSphere61
   \Express\profiles\default\logs\DefaultDB_migrationLogSun-Dec-18-13.31.40-CST-2005.log
   ```

   Otherwise, the log displays error messages in the format of the following example:

   ```
   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB>

   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB> took 0.26 seconds

   creating target db <jdbc:derby:/opt/WebSphere61/Express/profiles/default/databases
   /_opt_WebSphere51_AppServer_bin_DefaultDB>

    ERROR: An error occurred during migration. See debug.log for more details.
   ```

```
shutting down databases

shutting down databases took   0.055 seconds
```

5. For more data about a migration error, consult the debug log that corresponds with the database migration log. The WebSphere Application Server migration utility triggers a *debug migration trace* by default; this trace function generates the database debug logs. The full path name of a debug log is app_server_root/profiles/*profileName*/logs/*myFulldbPathName*_migrationDebug*timestamp*.log.

   The following lines are a sample of debug text. The lines display detailed exception data for the error that is referenced in the previous sample of database migration log data.

```
java.sql.SQLException: Database_opt_WebSphere51_AppServer_bin_DefaultDB already exists. Aborting migration
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.doMigrate(Unknown Source)
 at com.ibm.db2j.tools.MigrateFrom51.doMigrate(Unknown Source)
 at com.ibm.ws.adapter.migration.CloudscapeMigrationUtility.migr
```

Results

- The migration utility for the application server changes your Apache Derby or Cloudscape JDBC configurations whether or not it successfully migrates the database instances that are accessed by your applications. The tool changes the class paths for the Derby or Cloudscape JDBC provider, data source implementation classes, and data source helper classes. The following table depicts these changes:

Table 1. New class information. The following table depicts those changes mentioned in the previous section.

Class type	Old value	New value
JDBC provider class path	`${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar`	`${DERBY_JDBC_DRIVER_PATH}/derby.jar` • Where `DERBY_JDBC_DRIVER_PATH` is the WebSphere environment variable that defines your Cloudscape JDBC provider • Where `derby.jar` is the base name of the JDBC driver class file (In your environment, reference the JDBC driver class file by the full path name.)
Data source implementation class: Connection pool	`com.ibm.db2j.jdbc.DB2jConnectionPool DataSource`	`org.apache.derby.jdbc.EmbeddedConnection PoolDataSource`
Data source implementation class: XA	`com.ibm.db2j.jdbc.DB2jXADataSource`	`org.apache.derby.jdbc.EmbeddedXADataSource`
Data source helper class	`com.ibm.websphere.rsadapter.Cloudscape DataStoreHelper`	`com.ibm.websphere.rsadapter.Derby DataStoreHelper`

  Additionally, the db2j.properties file changes:
  - The name app_server_root/cloudscape/dbj.properties changes to app_server_root/derby/derby.properties
  - Within the file, property names change from db2j.drda.* to derby.drda.*
- A partial or a completely successful database migration changes the location and name of the database according to the following example:
  - **Old database name:** `c:\temp\`*mydb*
  - **New database name:** The new name includes a hash code that combines the entire path name of the old database and the migration time stamp. The new name also includes the old database name and time stamp exactly. For example:

    app_server_root\profiles\profile_name\databases*my_database*_hashCode_timestamp

  Note: For both partial and failed migrations, the log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

What to do next

If you experience a partial migration, attempt to troubleshoot the Cloudscape or Derby database only if you have expert knowledge of these database types. Otherwise, delete the new database. Perform the manual

migration procedure on the original database, just as you do for each database that completely fails auto-migration. See the topic, Upgrading Cloudscape manually, for instructions.

For successfully migrated Derby or Cloudscape instances, be aware that new cell-scoped data sources can only be used by nodes that run version 6.0.2 or later of the application server. Earlier versions of the product do not support the new versions of Derby or Cloudscape. When applications on nodes that are earlier than version 6.0.2 try to access a Cloudscape or Derby data source, the application server will issue exceptions at run time.

After a successful database migration, reboot the database and compress tables to improve performance. See the Apache Derby documentation for instructions.

Upgrading Cloudscape manually

During the upgrade of your application server, the migration tool attempts to upgrade instances of Cloudscape that are accessed through the embedded framework only. The automatic upgrade excludes Cloudscape instances that transact with applications through the Network Server framework. This exclusion eliminates the risk of corrupting third party applications that access the same database instances as the application server. You must manually upgrade database instances that are accessed through the Network Server framework. Do the same for databases that fail the automatic migration.

Before you begin

Do not use Apache Derby Version 10.1.x or Cloudscape v10.1.x as a production database. Use them for development and test purposes only.

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.

For instances of Cloudscape that are accessed through the embedded framework, determine which instances completely failed the automatic upgrade process and which ones were only partially upgraded. The topic on verifying the Cloudscape automatic migration documents how to uncover database errors and diagnostic data from various migration logs. The log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

To minimize the risk of migration errors for databases that were only partially upgraded during the automatic migration process, delete the new database. Troubleshoot the original database according to the log diagnostic data, then perform manual migration on the original database.

About this task

The following section consists of steps to migrate Cloudscape instances that are accessed through both the embedded framework as well as the Network Server framework. Steps that apply only to the Cloudscape Network Server framework are marked accordingly. As a migration best practice, ensure that your user ID has one of the following authorities:

- Administrator of the application server that accesses the Cloudscape instance
- A umask that can access the database instance

Otherwise, you might see runtime errors about the database instance being read-only.

Procedure

1. **Network Server framework only:** Ensure that every client of the Cloudscape database can support Cloudscape v10.1.x or Apache Derby. Application server clients of the database must run versions 6.02.x or later of the application server.

In the case of mixed-node cells, remember that only nodes of Version 6.02 or later of the application server can use data sources that you create post-migration for access to Cloudscape 10.1.x or Apache Derby. Earlier versions of the product do not support Cloudscape 10.1x or Apache Derby. When applications on nodes that are earlier than v6.02 try to access a cell-scoped Cloudscape 10.1.x or Apache Derby data source, the application server will issue exceptions at run time.

2. **Network Server framework only:** Take the database offline. No clients can access it during the migration process.
3. Examine a sample Cloudscape migration script that the application server provides, either db2jmigrate.bat or db2jmigrate.sh. The path of both scripts is *app_server_root*\derby\bin\embedded. You can modify the script according to the requirements of your environment. Consult the Cloudscape migration document for information about options that you can use with the script. For example, you can use the following option to specify the DDL file for the new database:

   `-DB2j.migrate.ddlFile=`*`filename`*
4. To generate database debug logs when you run the migration script, ensure that the debug migration trace is active. By default, this trace function is enabled. Reactivate the debug trace if it is disabled.
   a. To set the trace options in the administrative console, click **Troubleshooting** > **Logging and Tracing** in the console navigation tree.
   b. Select the application server name.
   c. Click **Change Log Level Details**.
   d. Optional: If **All Components** has been enabled, you might want to turn it off, and then enable specific components.
   e. Optional: Select a component or group name. For more information see the topic on log level settings. If the selected server is not running, you will not be able to see individual component in graphic mode.
   f. Enter a trace string in the trace string box. In this case, enter one of the following:
      - all traces*=all
      - com.ibm.ws.migration.WASUpgrade=all

      For more information on tracing read the topic on working with trace.
   g. Select **Apply**, then **OK**.
5. Specify your old database name and the full post-migration path of the new database name when you run the script. For example: `E:\WebSphere\AppServer\derby\bin\embedded>db2jMigrate.bat` *`myOldDB myNewDB`* The logs from the automatic migration provide the exact path names to specify for both the old database and the target database. You must use this target database name to specify the new database, because your migrated Cloudscape data sources (updated by the WebSphere Application Server migration utilities) now point to the target database name. The following sample text demonstrates how log messages display target database names:

   ```
   DSRA7600E: Cloudscape migration of database instance C:\temp\migration2\profiles\AppSrv01\
   installedApps\ghongellNode01Cell\DynamicQuery.ear\EmployeeFinderDB to new database instance
   C:\WebSphere\AppServer\profiles\AppSrv01\databases\C__WAS602_AppServer_profiles_AppSrv01_
   installedApps_ghongellNode01Cell_DynamicQuery.ear_EmployeeFinderDB failed,
   reason: java.sql.SQLException: Failure creating target db
   ```

   For instances of Cloudscape that are accessed through the Network Server framework, input any name that you want for the new database. Remember to modify your existing data sources to point to the new database name.
6. When the migration process ends, examine the database migration log to verify the results. The path name of each database migration log is *app_server_root*/logs/derby/*myFulldbPathName*_migrationLog.log.

   For a successful migration, the database migration log displays a message that is similar to the following text:

```
Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
Migration Completed Successfully
E:\WebSphere\AppServer\derby\bin\embedded>
```

Otherwise, the log displays error messages in the format of the following example:

```
Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
ERROR: An error occurred during migration. See debug.log for more details.
ERROR XMG02: Failure creating target db
java.sql.SQLException: Failure creating target db
        at com.ibm.db2j.tools.migration.MigrationState.getCurrSQLException(Unknown Source)
        at com.ibm.db2j.tools.migration.MigrateFrom51Impl.handleException(Unknown Source)
        at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
        at com.ibm.db2j.tools.migration.MigrateFrom51Impl.main(Unknown Source)
        at com.ibm.db2j.tools.MigrateFrom51.main(Unknown Source)
```

...

7. For more data about a migration error, consult the debug log that corresponds with the database migration log. The full path name of a debug log file is *app_server_root*/logs/derby/*myFulldbPathName*_migrationDebug.log.

   The following lines are a sample of debug text.

```
sourceDBURL=jdbc:db2j:E:\WebSphere\myOldDB
 newDBURL=jdbc:derby:e:\tempo\myNewDB
 ddlOnly=false
connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB>
connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB> took   0.611 seconds
creating target db <jdbc:derby:e:\tempo\myNewDB>
creating target db <jdbc:derby:e:\tempo\myNewDB> took   6.589 seconds
initializing source db data structures
initializing source db data structures took   0.151 seconds
recording DDL to create db <E:\WebSphere\myOldDB>
recording DDL to create db <E:\WebSphere\myOldDB> took   5.808 seconds
```

Results

As indicated in the previous steps, the database migration log displays either a `Migration Completed Successfully` message, or a message containing migration failure exceptions.

What to do next

- For databases that fail migration, troubleshoot according to the logged error data. Then rerun the migration script.
- To access successfully upgraded databases through the embedded framework, modify your data sources to point to the new database names.
- To access successfully upgraded databases through the Network Server framework, you can use either the DB2® Universal JDBC driver or the Derby Client JDBC driver.
  - If you want your existing JDBC configurations to continue to use the DB2 Universal JDBC driver, modify your data sources to point to the new database names.
  - If you want to use the Derby Client JDBC driver, which can support XA data sources, modify your JDBC providers to use the new Derby Client JDBC driver class and the new data source implementation classes. Then reconfigure every existing data source to use the correct Derby data source helper class, and to point to the new database name.

    Consult the article in the information center on vendor-specific data sources minimum required settings for all of the new class names.

Directory conventions

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

Default product locations - z/OS

app_server_root
Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS®.

The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
Refers to the installation root directory for Web Server Plug-ins.

profile_root
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are *server.root* and *user.install.root.*

In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

The corresponding product variable is *smpe.install.root*.

The default is /usr/lpp/zWebSphere/V8R0.

Chapter 5. Migrating 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.

Migrating servers from multi-broker replication domains to data replication domains

You can migrate multi-broker replication domains to data replication domains. Any multi-broker domains that exist in your application server environment were created with a previous version of the product.

Before you begin

Determine if the application server configuration you are migrating:

1. Uses an instance of data replication service in peer-to-peer mode or in client/server mode.

   Before you begin migrating a client/server mode replication domain, consider if migrating your replication domains might cause a single point of failure. Because you migrate the servers to the new type of replication domain one at a time, you risk a single point of failure if there are 3 or fewer application servers. Before migrating, configure at least 4 servers that use multi-broker replication domains. Perform the following steps to migrate the multi-broker domains to data replication domains:

   Dynamic cache replication domains use the peer-to-peer topology.
2. Uses HTTP session memory-to-memory replication that is overloaded at the application or web module level.

   If the application server configuration you are migrating uses HTTP session memory-to-memory replication that is overloaded at the application or web module level, you must upgrade your deployment manager to the current version of the product before you start the migration process.

About this task

After you upgrade your deployment manager to the latest version of the product, you can only create data replication domains. Any multi-broker domains that you created with a previous version of the product are still functional, however, you cannot use the administrative console to create new multi-broker domains or replicators.

The different versions of application servers cannot communicate with each other. When migrating your servers to the current version of the product, keep at least two application servers running on the previous version so that replication remains functional.

Make sure that all of your application servers that are using this multi-broker domain have been migrated to the current version of the product before you start to migrate any multi-broker domains that exist in your configuration.

To migrate the multi-broker domains that exist in your configuration:

Procedure

1. Migrate two or more of your existing servers to the current version of the product. The remaining servers on the previous version of the product can still communicate with each other, but not with the migrated servers. The migrated servers can also communicate with each other.

© Copyright IBM Corp. 2011

2. In the administrative console, create an empty data replication domain. Click **Environment > > Replication domains > New** to create an empty data replication domain.
3. Add two of your migrated servers to the new data replication domain.

   For example, if you are migrated four servers, only add two of them to the new replication domain.
4. Configure the two servers as consumers of the replication domain.

   Configuring the servers as consumers of the replication domain enables them to use the new domain to share data.
5. Add some of the clients to the new data replication domain.

   Perform this step only if the application server configuration you are migrating uses an instance of data replication service in client/server mode.
6. Configure these clients as consumers of the replication domain.
7. Verify that the new data replication domain are successfully sharing data.

   Only the servers and clients that are added to the data replication domain and are configured as consumers of this domain can use the data replication domain functions.
8. Add the rest of your migrated servers to the new data replication domain.

   When the servers can use the new data replication domain to successfully share data, migrate the rest of the servers that are using the multi-broker replication domain to the new data replication domain.

   For example, if you are migrated four servers, add the remaining two servers to the new replication domain.
9. Configure these servers as consumers of the replication domain.
10. Add the rest of the clients to the new data replication domain.

    Perform this step only if the application server configuration you are migrating uses an instance of data replication service in client/server mode.
11. Configure these clients as consumers of the replication domain.
12. Restart all of the application servers and clients.
13. Delete the empty multi-broker replication domain.

What to do next

During this process, you might lose existing sessions. However, the application remains active through the entire process, so users do not experience down time during the migration. Create a new replication domain for each type of consumer. For example, create one replication domain for the session manager and another replication domain for dynamic cache.

Chapter 6. Migrating 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.

Migrating enterprise bean code from Version 1.1 to Version 2.1

Enterprise JavaBeans (EJB) Version 2.1-compliant beans can be assembled only in an EJB 2.1-compliant module, although an EJB 2.1-compliant module can contain a mixture of Version 1.x and Version 2.1 beans.

About this task

The EJB Version 2.1 specification mandates that prior to the EJB container starting a findBy*Method* query, the state of all enterprise beans that are enlisted in the current transaction be synchronized with the persistent store. (This action is so the query is performed against current data.) If Version 1.1 beans are reassembled into an EJB 2.1-compliant module, the EJB container synchronizes the state of Version 1.1 beans as well as that of Version 2.1 beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.

The following information generally applies to any enterprise bean that currently complies with Version 1.1 of the EJB specification. For more information about migrating code for beans produced with the Rational Application Developer tool, see the documentation for that product.

Procedure

1. In beans with container-managed persistence (CMP) version 1.x, replace each CMP field with abstract get and set methods. In doing so, you must make each bean class abstract.
2. In beans with CMP version 1.x, change all occurrences of `this.`*`field`* `=` *`value`* to `set`*`Field`*`(`*`value`*`)`.
3. In each CMP bean, create abstract get and set methods for the primary key.
4. In beans with CMP version 1.x, create an EJB Query Language statement for each finder method.

   Note: EJB Query Language has the following limitations in Application Developer Version 5:
   - EJB Query Language queries involving beans with keys made up of relationships to other beans appear as invalid and cause errors at deployment time.
   - The IBM EJB Query Language support extends the EJB 2.1 specification in various ways, including relaxing some restrictions, adding support for more DB2 functions, and so on. If portability across various vendor databases or EJB deployment tools is a concern, then care should be taken to write all EJB Query Language queries strictly according to instructions described in Chapter 11 of the EJB 2.1 specification.
5. In finder methods for beans with CMP version 1.x, return java.util.Collection instead of java.util.Enumeration.
6. Update handling of non-application exceptions.
   - To report non-application exceptions, throw javax.ejb.EJBException instead of java.rmi.RemoteException.
   - Modify rollback behavior as needed: In EJB versions 1.1 and 2.1, all non-application exceptions thrown by the bean instance result in the rollback of the transaction in which the instance is running; the instance is discarded. In EJB 1.0, the container does not roll back the transaction or discard the instance if it throws java.rmi.RemoteException.
7. Update rollback behavior as the result of application exceptions.

© IBM Corporation 2009

- In EJB versions 1.1 and 2.1, an application exception does not cause the EJB container to automatically roll back a transaction.
- In EJB Version 1.1, the container performs the rollback only if the instance has called setRollbackOnly() on its EJBContext object.
- In EJB Version 1.0, the container is required to roll back a transaction when an application exception is passed through a transaction boundary started by the container.

8. Update any CMP setting of application-specific default values to be inside ejbCreate (not using global variables, since EJB 1.1 containers set all fields to generic default values before calling ejbCreate, which overwrites any previous application-specific defaults). This approach also works for EJB 1.0 CMPs.

Migrating enterprise bean code to the supported specification

Support for the Enterprise JavaBeans (EJB) 3.1 specification is added for this product.

Before you begin

There are no migration issues associated with using EJB 3.x beans. Existing applications continue to run as-is and compile without error.

Note: The EJB 3.0 and EJB 3.1 specifications have deprecated the use of EJB 1.1 style entity beans. While using EJB 2.x and earlier modules in the product has not yet been deprecated, you are encouraged to start migrating to Java Persistence API (JPA) or JDBC.

About this task

Follow these steps as appropriate for your application deployment.

Procedure

1. Modify enterprise bean code for changes in the specification.
   - You must migrate the Version 1.1 beans to Version 2.x beans and redeploy them on the product.

     Note: The EJB Version 2.0 specification mandates that before the EJB container runs a findBy*Method* query, the state of all enterprise beans enlisted in the current transaction be synchronized with the persistent store. This synchronization is done so that the query is performed against current data. When Version 1.1 beans are reassembled into an EJB 2.x-compliant module, the EJB container synchronizes the state of Version 1.1 beans, as well as that of Version 2.x beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.
2. Reassemble and redeploy all modules to incorporate migrated code.

Adjusting exception handling for EJB wrapped applications migrating from version 5 to version 7

Because of a change in the Java APIs for XML based Remote Procedure Call (JAX-RPC) specification, Enterprise JavaBeans (EJB) applications that could be wrapped in WebSphere Application Server Version 5.1 cannot be wrapped in version 6 or 7 unless you modify the code to the exception handling of the base EJB application.

About this task

Essentially, the JAX-RPC version 1.1 specification states:

```
a service specific exception declared in a remote method signature must be a
checked exception. It must extend java.lang.Exception either directly or indirectly
but it must not be a RuntimeException.
```

So it is no longer possible to directly use java.lang.Exception or java.lang.Throwable types. You must modify your applications using service specific exceptions to comply with the specification.

Procedure

1. Modify your applications that use service specific exceptions. For example, say that your existing EJB uses a service specific exception called UserException. Inside of UserException is a field called *ex* that is type java.lang.Exception. To successfully wrapper your application with web services in WebSphere Application Server version 7, you must change the UserException class . In this example, you could modify UserException to make the type of *ex* to be *java.lang.String* instead of java.lang.Exception.

```
new UserException class:

package irwwbase;

/**
 * Insert the type's description here.
 * Creation date: (9/25/00 2:25:18 PM)
 * @author: Administrator
 */


public class UserException extends java.lang.Exception {

       private java.lang.String _infostring = null;
       private java.lang.String ex;
/**
 * UserException constructor comment.
 */

public UserException() {
       super();
}
/**
 * UserException constructor comment.
 */
public UserException (String infostring)
{
       _infostring = infostring;
} // ctor
/**
 * Insert the method's description here.
 * Creation date: (11/29/2001 9:25:50 AM)
 * @param msg java.lang.String
 * @param ex java.lang.Exception
 */
public UserException(String msg,String t) {
       super(msg);
       this.setEx(t);

       }
       /**
        * @return
        */
       public java.lang.String get_infostring() {
              return _infostring;
       }

       /**
        * @return
        */
       public java.lang.String getEx() {
              return ex;
       }

       /**
        * @param string
```

```
         */
        public void set_infostring(java.lang.String string) {
               _infostring = string;
        }

        /**
         * @param Exception
         */
        public void setEx(java.lang.String exception) {
               ex = exception;
        }

        public void printStackTrace(java.io.PrintWriter s) {
          System.out.println("the exception is :"+ex);
          }

}
```

2. Modify all of the exception handling in the enterprise beans that use it. You must ensure that your enterprise beans are coded to accept the new exceptions. In this example, the code might look like this:

```
new EJB exception handling:

try {
      if (isDistributed()) itemCMPEntity = itemCMPEntityHome.findByPrimaryKey(ckey);
      else itemCMPEntityLocal = itemCMPEntityLocalHome.findByPrimaryKey(ckey);
  } catch (Exception ex) {
       System.out.println("%%%%% ERROR: getItemInstance - CMPjdbc " + _className);
       ex.printStackTrace();
       throw new UserException("error on itemCMPEntityHome.findByPrimaryKey(ckey)",ex.getMessage());
   }
```

Chapter 7. Migrating 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 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 resource adapter or the ASF component of the JMS Version 1.0.2 specification.

Migrating from WebSphere Application Server Version 5 embedded messaging

This set of topics describes the migration of JMS applications from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in later versions.

About this task

You can temporarily use JMS resources developed for WebSphere Application Server Version 5.1 with service integration in later versions, however you should ideally migrate all resources to the later versions as soon as possible.

For migration strategies, see “Migration from Version 5 embedded messaging” on page 26.

When migrating a Version 5.1 node to a later version, you do not have to modify your JMS applications; they can continue to use most of the existing Version 5.1 JMS deployment, installation, and configuration settings. However, when migrating a Version 5.1 message-driven bean (MDB) application, you must modify the configuration to provide a JMS activation specification, not a listener port (MDBs do not use listener ports), then redeploy or reinstall the MDB application.

For more information about migrating from WebSphere Application Server Version 5 embedded messaging, see the following links:

- “Migrating a stand-alone application server from Version 5 embedded messaging” on page 36
- “Migrating a network deployment configuration from Version 5 embedded messaging” on page 39
- “Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 1” on page 42
- “Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2” on page 47

© Copyright IBM Corp. 2011

- “Migration from Version 5 embedded messaging”
- “Migrating Version 5.1 messages using the message migration utility” on page 30

Migration from Version 5 embedded messaging

When you migrate from WebSphere Application Server Version 5 embedded messaging to the default messaging provider, you do not have to change Version 5.1 JMS applications, JMS resource definitions, or client JMS resource definitions. Use of bus resources is enabled by a WebSphere MQ client link, and the wildcard syntax is converted automatically for interoperation between Version 5.1 and later versions.

- “You do not have to change Version 5.1 JMS applications”
- “You do not have to change Version 5.1 JMS resource definitions”
- “You do not have to change Version 5.1 client JMS resource definitions” on page 27
- “Before migrating a WebSphere Application Server Version 5.1 node, you might have to consume messages” on page 27
- “You should replace MDB listener ports with JMS activation specifications” on page 28
- “Use of bus resources in later versions is enabled by a WebSphere MQ client link” on page 28
- “The wildcard syntax is converted automatically for interoperation between Version 5.1 and later versions” on page 29
- “Configuration scripts for WebSphere Application Server Version 5 embedded messaging should not be run, and fail if they are run” on page 29

You do not have to change Version 5.1 JMS applications

When migrating a WebSphere Application Server Version 5.1 node to later versions, you do not need to make any changes to JMS applications; they can continue to use their same deployment and installation, and their same configurations of Version 5.1 JMS resources (with one exception given in “You do not have to change Version 5.1 JMS resource definitions”).

The applications can continue to use the same JMS API classes and listener ports, but rather than communicating with a WebSphere MQ queue manager, the applications communicate with a messaging engine on a service integration bus.

You do not have to change Version 5.1 JMS resource definitions

When migrating a WebSphere Application Server Version 5.1 node to later versions, you do not need to make any changes to JMS resource definitions. JMS applications can continue to use their same configurations of Version 5.1 JMS resources, with the following exception.

The exception to this is for JMS applications that use the Version 5 embedded messaging provider DIRECT port for publish/subscribe messaging, as set on the WebSphere Application Server topic connection factory. If any WebSphere Application Server Version 5.1 topic connection factory has the Port property set to DIRECT, change it to QUEUED before use with the default messaging provider in the later version.

If a node is migrated to a later version, the JMS resources are not changed, except to use the new naming convention. That is, the administrative name for the Version 5 embedded messaging JMS resources is changed from **WebSphere JMS Provider** resources to **V5 Default Messaging** resources.

Listener ports are copied over unmodified from the Version 5.1 configuration, and are used on the later version whenever the associated Version 5 default messaging resources are used.

After migration to the later version, the JMS resources are implemented through the default messaging provider. You can use the administrative console to manage the JMS resources as Version 5.1 default messaging JMS resources. For example, in the administrative console you can list Version 5.1 default

messaging JMS queue connection factories by clicking **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queue connection factories**

You should replace Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources as soon as is conveniently possible (after all JMS applications that use those resources have been moved onto the later version). This enables you to benefit from the better performance of the default messaging provider, and to exploit the use of multiple messaging engines in a service integration bus, and other default messaging functions enabled by service integration technologies.

You can replace JMS resources manually, for example by using the WebSphere Application Server administrative console. Alternatively, you can replace the resources programmatically, for example by some scripting that retrieves the Version 5.1 property values then creates JMS resources for the later version, with values appropriate to that environment and your use of the Version 5.1 properties.

New JMS resources should be created as JMS resources in the later version. Any wsadmin or JMX scripts that create Version 5.1 JMS resources must be changed to create JMS resources for use in the later version.

You do not have to change Version 5.1 client JMS resource definitions

JMS client applications that have JMS resources configured by using the Application Client Resource Configuration Tool (ACRCT) in WebSphere Application Server Version 5.1 should continue to work without change with later versions.

You should replace Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources as soon as is conveniently possible (after all the application servers have been migrated onto the later version). You should use the ACRCT ofo the later version to replace the JMS resources and to create any new JMS resources. New JMS resources should be created as JMS resources for the later version.

Before migrating a WebSphere Application Server Version 5.1 node, you might have to consume messages

When migrating a WebSphere Application Server Version 5.1 node to a later version, any messages (and knowledge of durable subscriptions) held by the JMS server are not migrated automatically. Whether you have to drain all the JMS queues depends upon your migration strategy.

You should use one of the following migration strategies:

- Before migrating a WebSphere Application Server Version 5.1 node, you stop Version 5.1 JMS applications using the JMS queues that are to be migrated.
  - Stop all message-producing JMS applications in the WebSphere Application Server Version 5.1 environment. For example, you can use the administrative console to stop the applications, as described in Starting or stopping enterprise applications.
  - Allow all message-consuming JMS applications (including those consuming publications as a result of durable subscriptions) to continue until all the JMS queues are drained, then stop those applications.

  For each JMS queue defined on the JMS server, the migration process automatically creates a new bus queue with the same name as the Version 5.1 JMS queue, and creates a message point assigned to the messaging engine. Messages sent to the JMS queues are stored and processed at the message point.
- Leave the JMS server node at WebSphere Application Server Version 5.1 and write an application to run on a later-level node, to access the Version 5.1 JMS server, get the Version 5.1 messages, and resend or publish them to applications on the later version.

- Use the message migration utility to migrate the contents of existing WebSphere Application Server Version 5 embedded messaging queues, as described in “Migrating Version 5.1 messages using the message migration utility” on page 30.

  Note: the message migration utility does not migrate publish/subscribe messages or durable subscriptions.

You should replace MDB listener ports with JMS activation specifications

A JMS application that uses a message-driven bean and its listener port in WebSphere Application Server Version 5.1 can continue to use the listener port without change in later versions. However, the message listener service uses the Application Server Facilities (ASF), which are an optional part of the JMS specification. Also, ASF is not supported by the service integration technologies on which the later default messaging provider is implemented.

The default messaging provider in later versions is implemented as a Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) resource adapter, for which inbound connectivity is configured as an activation specification. Therefore, as soon as is conveniently possible, you should replace any listener port with a JMS activation specification for use by MDB applications with the default messaging provider.

If you also classify inbound MDB work requests for the z/OS workload manager, you should convert the InboundClassification type="mdb" classifications with SibClassifications type="jmsra" classifications. For more information about classifying MDB work for the default messaging provider, see Workload classification file.

If your inbound MDB work is for another JMS provider, such as WebSphere MQ, you should continue to use any listener ports and their associated InboundClassification type="mdb" classifications for the z/OS workload manager. You cannot create specific z/OS workload manager classifications for inbound MDB work to JCA resource adapters other than the later default messaging provider - such MDB work uses the default z/OS workload manager classification. For more information about classifying MDB work for listener ports, see Workload classification file.

If you used the listener port retry count in WebSphere Application Server Version 5.1, there is one extra consideration. The Java EE Connection Architecture has no concept of a listener port retry count, so this is not supported by the later default messaging provider. This should not present a problem because the default messaging provides destinations with a “Maximum failed deliveries” setting. This defines the maximum number of times that the service tries to deliver a message to the destination before forwarding it to the exception destination. Although applications do not have to be changed, any wsadmin or JMX scripts that make use of the listener port retry count must be changed to make use of the “Maximum failed deliveries” setting for use in the later version.

Use of bus resources in later versions is enabled by a WebSphere MQ client link

Version 5 embedded messaging uses WebSphere MQ technology, and for communication uses WebSphere MQ client protocols. The default messaging provider in later versions uses fully-integrated Java technology, and JMS applications access JMS resources through the messaging engines on a service integration bus.

JMS applications developed for WebSphere Application Server Version 5.1 can use resources on a service integration bus through a *WebSphere MQ client link* assigned to a messaging engine on the service integration bus. The WebSphere MQ client link is provided only for use with JMS applications developed for WebSphere Application Server Version 5.1. This link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by JMS applications developed for WebSphere Application Server Version 5.1 and the protocols used by messaging engines in the later version. This link can be used to access JMS resources that have a destination anywhere on the bus or on any other connected bus.

If you migrate a WebSphere Application Server Version 5.1 node to a later version, a default WebSphere MQ client link is created automatically for the node, and assigned to the messaging engine for the application server that is also created by the migration process. The default link has the following properties:

Property	Value
Name	Default.MQClientLink
Description	Default MQ Client Link
Queue manager name	WAS_*nodeName*_jmsserver
Channel name	WAS.JMS.SVRCONN

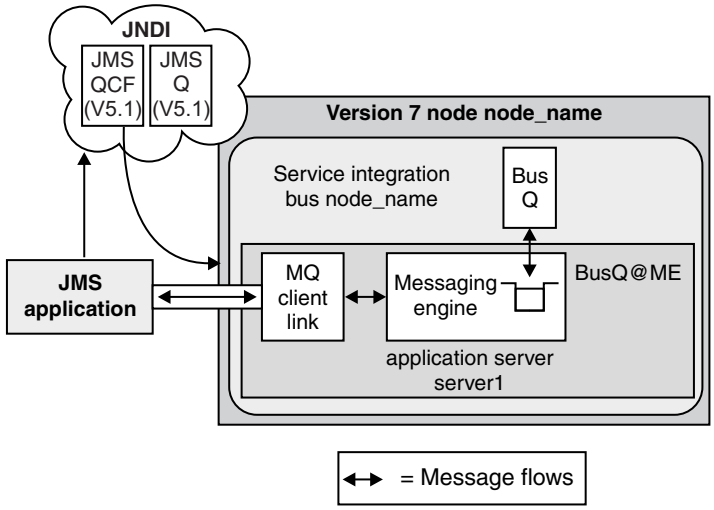


Figure 1. WebSphere Application Server 5 JMS application scenario after migration

This figure shows an example single-node scenario after migrating the node to WebSphere Application Server Version 7.0. The JMS resources are now managed as Version 5 embedded messaging JMS resources implemented by the Version 7.0 default messaging provider. Also, a WebSphere MQ client link and bus queue have been created and assigned to the messaging engine, to enable JMS applications developed for WebSphere Application Server Version 5.1 to use the JMS resources.

The wildcard syntax is converted automatically for interoperation between Version 5.1 and later versions

Existing WebSphere Application Server Version 5.1 client applications using Version 5.1 connection factory and destination definitions use the WMQI wildcard convention. Such applications can connect to the default messaging provider and service integration bus, and automatically have their wildcard syntax mapped to the XPath convention when subscriptions are created. Any display of these subscriptions through the administrative interface of a later version shows the XPath syntax.

Configuration scripts for WebSphere Application Server Version 5 embedded messaging should not be run, and fail if they are run

When upgrading from WebSphere Application Server Version 5.1, the Version 5 embedded messaging configuration scripts are retained, but have no influence on the default messaging provider in the later version, and fail if run. **Do not run the scripts.** Failure of the scripts can appear to be a failure of the installation of the later version, but no damage or action to the installation results from running the scripts.

Migrating Version 5.1 messages using the message migration utility

This set of topics describes the migration of WebSphere Application Server Version 5.1 messages to later version messages, using the WebSphere Application Server message migration utility.

About this task

The message migration utility takes the contents of existing WebSphere Application Server Version 5 embedded messaging queues and puts them on the new default messaging queues in the later version.

Note: the message migration utility does not migrate publish/subscribe messages or durable subscriptions.

Migrating Version 5 embedded messages, by using the message migration utility, is described in more detail in the following topics:

- "WebSphere Application Server message migration utility"
- "Installing the message migration utility"
- "Running the message migration utility" on page 31
- "Reversing the migration of messages by using the message migration utility" on page 32
- "XA recovery" on page 33
- "Migration of message fields" on page 34

WebSphere Application Server message migration utility

The message migration utility takes the contents of existing WebSphere Application Server Version 5 embedded messaging queues and puts them onto service integration bus messaging queues.

You run the message migration utility only once for a particular queue unless one of the following is true:

1. The first message migration attempt was unsuccessful.
2. You want to reverse the message migration (that is, move the messages back to WebSphere Application Server Version 5.1).

If you migrate a server and messages from Version 5.1 to a later version and then decide to reverse the migration, the reversal must be done in the following order:

1. Run the message migration utility in the direction: later version to Version 5.1.
2. Reverse the server migration.

It is possible to migrate multiple queues during a single execution of the utility.

Installing the message migration utility

How to install the WebSphere Application Server message migration utility, including the steps that are necessary before the installation.

Before you begin

Before you install the message migration utility ensure that you have backed up your embedded messaging queues on WebSphere Application Server Version 5.1. For details of backing up queues refer to the WebSphere MQ System Administration Guide. This guide applies to backing up WebSphere MQ queues but can be used to back up your embedded messaging queues.

Note: If you back up a WebSphere Application Server Version 5.1 queue before migration and then reinstate the whole queue and start migrating the same queue again, you will get two copies of the messages in the version you migrated to.

About this task

To install and start the message migration utility, use the administrative console to complete the following steps.

Procedure

1. Install the enterprise application:
   a. Click **Applications -> New Application -> New Enterprise Application**.
   b. Specify the location of the SibMsgMigrationUtility.ear file and click **Next**. This file is located in the installableApps directory.
   c. Accept all the defaults on the next screen and click **Next**.
   d. You do not have to change any other settings, so click on the final step on the left **Summary**, then click **Finish** to install the message migration utility.
   e. When you see **Application: WebSphere Message Migration Utility installed successfully**, save the changes to the master configuration.
2. Start the application:
   a. Click **Applications -> Application Types -> WebSphere enterprise applications**.
   b. Select **WebSphere Message Migration Utility**.
   c. Click **Start**.
3. Check the port number. Providing the port number used by the web container has not been changed you can go to the following address using a web browser and the tool will start: http://<yourhostname>:9080/MessageMigrationUtility. If the port number was changed, replace "9080" with the correct port.

Running the message migration utility

The WebSphere Application Server message migration utility uses an XA (globally coordinated) transaction to migrate messages from the V5 embedded messaging provider on a WebSphere Application Server Version 5.1 application server to the default messaging provider (service integration) on an application server at a later version of the product.

Before you begin

1. Ensure that you have a WebSphere Application Server Version 5.1 system that contains an embedded messaging server.

   The Version 5.1 server must be started.
2. Ensure that no applications are reading from the WebSphere Application Server Version 5.1 queue manager when you run the message migration utility.
3. Ensure that you have not modified or deleted the message queues on the Version 5.1 application server.
4. Ensure that the version you are migrating to is a running system that fulfills the following conditions.
   a. The system is on the same host as the WebSphere Application Server Version 5.1 system.
   b. The system contains a messaging engine on the bus to which the messages will migrate (this is automatically included for you if you upgrade your server from Version 5.1 to a later version).

   Note: These conditions are met if you run the message migration utility at the correct point in the WebSphere Application Server migration sequence. The correct point is after you have run the WASPostUpgrade command and restarted the node on the version you migrate to.

About this task

You run the message migration utility only once for a particular queue, unless a failure occurs.

If a failure occurs during message migration, it is safe to run the message migration utility again in the same direction because messages are moved rather than copied. Whether you successfully try a failing message migration again, or delete the message, the message ordering of the remaining messages is preserved on the queue on the current version of the product.

During successful migration, messages are moved from the WebSphere Application Server Version 5.1 system to the later system. No copy is left on the Version 5.1 server queue.

For more information about the XA transaction, see “XA recovery” on page 33.

Procedure

1. Follow the actions indicated by the message migration utility.
2. In the "Select the direction of migration" panel, select **Migrate messages from Version 5 to *later_version***.
3. Select a message reliability to apply to messages that are to be migrated. More details on the choice of reliability levels can be found in the Message Reliability Levels topic.
4. You can migrate multiple queues during a single run of the utility.

Results

1. If the first message migration attempt fails, run the migration utility again.
2. If a queue has a failing message, complete one of the following operations:
   a. Try the failing message again, in case there is a transient error.
   b. Delete the failing message and go on to the next message.
   c. Stop the queue that has the failing message and move on to the next queue.

What to do next

For information about the reversal of message migration, see Reversing the migration of messages by using the message migration utility.

Reversing the migration of messages by using the message migration utility

You can use the message migration utility to reverse migrate messaging from the default messaging provider (the service integration bus) on an application server at the current version of the product to the Version 5 embedded messaging provider on a Version 5.1 application server. Reverse migration is carried out under XA coordination.

Before you begin

You must reverse the migration of the messages before deleting the application server on the service integration bus.

About this task

You run the message migration utility only once for a particular queue, unless a failure occurs.

If a failure occurs during message migration, it is safe to run the message migration utility again in the same direction because messages are moved rather than copied. Whether you successfully try a failing message migration again, or delete the message, the message ordering of the remaining messages is preserved on the queue on the current version of the product.

If the reverse migration takes place after a partial migration from WebSphere Application Server Version 5.1 to a later version, in which only some of the messages were migrated, then the message ordering is not preserved. However, if the target queue in the Version 5.1 application server is empty when the migration takes place (as it is likely to be) then the message order is retained.

Messages that are reverse-migrated appear on the queue after those that were not migrated at the first attempt.

Procedure

1. Follow the actions indicated by the message migration utility.
2. On the "Select the direction of migration" panel choose **Reverse a previous migration**.

Results

As in normal migration, if a queue has a failing message you have the choice of:

1. Trying the failing message again, in case there is a transient error.
2. Deleting the failing message and going on to the next message.
3. Stopping the queue that has the failing message and moving on to the next queue.

XA recovery

When a message is migrated between WebSphere Application Server Version 5 embedded messaging and the service integration bus on a later version of the product, an XA transaction is used. This ensures that the movement of a single message is carried out as a single unit of work even though there are two discrete and separate resources involved. If a transaction is in an indoubt state, the transaction manager attempts to recover the transaction when the application server next restarts.

Before you begin

For XA recovery to complete successfully:

1. All the resources involved in the transaction must be available to the transaction manager.

   Note: This is not a problem for the default messaging provider (the service integration bus) in later versions of the product, because it starts when the server (and the transaction manager) start. However, Version 5 embedded messaging is not told that the server is starting, and the embedded queue manager needs to be started manually to allow XA recovery to complete.
2. The resources involved must be started in the same way. The queue manager must be started using the same startup parameters, for example, the same TCP/IP port number as before.

   Note: When a transaction is in an indoubt state, information about the resources is saved in the transaction log.

About this task

The XA transaction uses two-phase commit, which ensures that all resources participating in a transaction are asked to "prepare" to commit and then, when all the resources have indicated they are ready, the transaction commits. XA transactions are coordinated by a transaction manager, which informs the resources to prepare, commit or roll back.

There is one XA transaction for each migrating message, and the controlling transaction manager is part of the service integration bus, although you will probably be unaware that the transaction manager is involved.

It is possible for a transaction to be in an indoubt state. This can occur if the transaction manager has told the participants in a transaction to prepare and they have successfully done so, but an error has occurred when the final commit is attempted. This can be caused by a communications failure between the transaction manager and a transaction participant. The recovery process generally involves reconnecting to each of the participants and telling them to roll back the transaction.

When the message migration utility connects to the Version 5 embedded messaging server, it creates a log of the connection parameters that were used in your profile directory on the server at the later level. For example, if the profile in use is "Profile1", then the log file is located in the following directory:

app_server_root/profiles/Profile1/logs/message_migration_utility.log

where *app_server_root* is the root directory for the installation of WebSphere Application Server.

Note: The log file contains details of every run of the utility. If you have run the utility more than once make sure that you go to the last entry.

Sample contents of the log file:

```
------------------------------------------------------------------
Migration Utility was started: Tue Feb 22 14:20:49 GMT 2005
------------------------------------------------------------------
   Browser details   : Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.5)
            Gecko/20041107 Firefox/1.0
   Referrer          : null
   Query String      : null
   User              : your.server.name:9080

Connection to WebSphere Application Server Version 5
Embedded Messaging server was successful using the following properties:

   Queue Manager Name: WAS_Node01_server1
   Hostname          : your.server.name
   TCP/IP Port       : 21179
   SVRCONN Channel   : WAS.JMS.SVRCONN

Connection to WebSphere Application Server Version version_number
default messaging provider was successful using the following properties:

   SIBus Name        : bus1

Message migration will take place using the following settings:

   Queues            : [queue2, queue1]
   Direction         : Version 5.1 -> Version version_number
   version_number reliability    : ReliablePersistent
```

Procedure

1. Start the Version 5 embedded messaging queue manager.
   Contact your system administrator if you are unsure how start the queue manager.
2. Start the application server on the later version of the product, if it is not already started.

Results

XA recovery continues automatically.

Migration of message fields

When you use the WebSphere Application Server message migration utility to migrate a message from JMS format to service integration format, most fields are migrated unchanged. However, some fields are changed and some are not supported by service integration.

Changes to messages due to migration

All JMS message types migrate unchanged:

1. Message

2. TextMessage
3. MapMessage
4. StreamMessage
5. ObjectMessage
6. BytesMessage

User properties set on the message by an application are also unaltered.

The JMS message types provide header fields that **can** change as a result of migration:

Table 2. JMS Message Header fields. The first column of the table contains the list of the header field names. The second column indicates if the fields of the messages are migrated without any changes.

Header field name	State after migration
JMSMessageID	Unchanged
JMSCorrelationID	Unchanged
JMSDeliveryMode	Unchanged
JMSPriority	Unchanged
JMSTimestamp	Unchanged
JMSExpiration	Unchanged
JMSRedelivered	Can be reset as a result of the migration process.
JMSType	Unchanged
JMSDestination	The name of the destination is unaltered. Other properties of the destination are mapped to their equivalents in the later version, where possible.
JMSReplyTo	The name of the reply destination is unaltered, and assumed to exist on the bus to which the messages are being migrated. **Note:** • References to temporary queues or topics are migrated in the same way as for permanent reply destinations. It will not be possible to send reply messages to these destinations because they will not exist in the bus. • Topic reply-to destinations are assumed to be topics within the default topic space, which must exist for the reply message to be sent.

JMSX properties **can** also change as a result of migration.

Table 3. JMSX properties. The first column of the table contains the list of the JMSX property names. The second column indicates if the properties have changed or not supported by the service integration bus after the migration.

JMSX property name	State after migration
JMSXUserID	Unchanged
JMSXAppID	Unchanged
JMSXDeliveryCount	Can be reset as a result of the migration process.
JMSXGroupID	Unchanged
JMSXGroupSeq	Unchanged
JMSXProducerTXID	Not supported by service integration bus.
JMSXConsumerTXID	Not supported by service integration bus.
JMSXRcvTimestamp	Not supported by service integration bus.
JMSXState	Not supported by service integration bus.

The following JMS_IBM properties **do not** change as a result of migration.

Table 4. JMS_IBM properties. The first column of the table contains the list of the JMS_IBM property names. The second column indicates the state of the properties after the migration. The JMS_IBM properties do not change after migration.

JMS_IBM property name	State after migration
JMS_IBM_Report_*	Unchanged
JMS_IBM_MsgType	Unchanged
JMS_IBM_Feedback	Unchanged
JMS_IBM_Format	Unchanged
JMS_IBM_PutApplType	Unchanged
JMS_IBM_Encoding	Unchanged
JMS_IBM_Character_Set	Unchanged
JMS_IBM_PutDate	Unchanged
JMS_IBM_PutTime	Unchanged
JMS_IBM_Last_Msg_In_Group	Unchanged

Migrating a stand-alone application server from Version 5 embedded messaging

Migrate a stand-alone application server from WebSphere Application Server Version 5 embedded messaging for use with the default messaging provider in later versions.

Before you begin

Before starting this task you must stop all Version 5.1 JMS applications that are using the JMS queues you want to migrate:

- Stop all message-producing JMS applications in the WebSphere Application Server Version 5.1 environment. For example, use the administrative console to stop the JMS applications, as described in Starting or stopping enterprise applications.
- Allow all message-consuming JMS applications (including those JMS applications that are consuming published messages as a result of durable subscriptions) to continue, until all the queues are drained, then stop the JMS applications.

About this task

When migrating a WebSphere Application Server Version 5.1 stand-alone application server to later versions, you do not have to make any changes to JMS applications that can continue to use their same deployment and installation, and their same configurations of Version 5.1 JMS resources, apart from one exception which is described below. For a more detailed explanation, see “Migration from Version 5 embedded messaging” on page 26.

Before migrating, consider the stand-alone application server scenario shown in the following figure Figure 2 on page 37.

- The JMS application uses JNDI to look up the JMS resources in the WebSphere Application Server namespace.
- The JMS resources in this example are a JMS queue connection factory (shown as JMS QCF) and a JMS queue (shown as JMS Q).
- WebSphere Application Server Version 5 embedded messaging uses WebSphere MQ technology, and is implemented through a JMS server that runs as the jmsserver service of the application server. The JMS application uses WebSphere MQ client protocols to communicate with the JMS server.

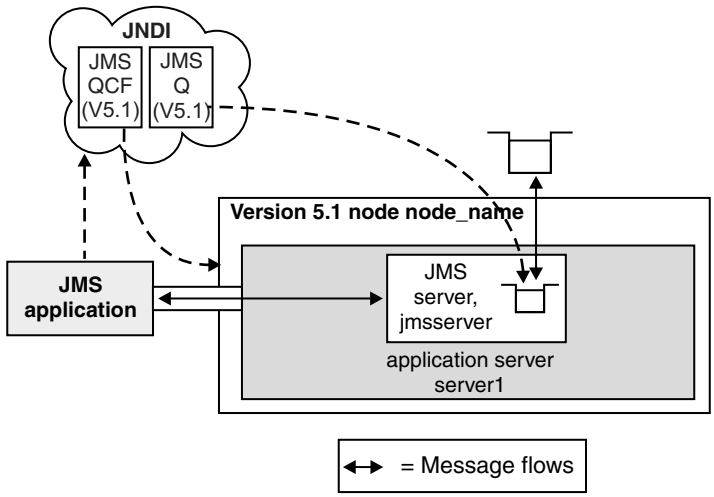


Figure 2. Stand-alone WebSphere Application Server 5 JMS application scenario before migration. This figure shows the example stand-alone application server scenario before migrating the stand-alone application server to a later version. The JMS application is supported by an application server. The JMS application can be running within the application server or as a JMS client application.

To migrate a stand-alone WebSphere Application Server environment from Version 5 embedded messaging to the default messaging provider in later versions, complete the following steps:

Procedure

1. Migrate the stand-alone WebSphere Application Server to the later version. See the documentation about migrating product configurations.

   As part of the task for migrating an application server, you should complete the following steps to continue using Version 5.1 default messaging JMS resources:

   a. Create a service integration bus, and add an application server to that bus.

      The default messaging provider is based on one or more service integration buses. JMS destinations are assigned to a service integration bus. To make use of resources through service integration technologies, you can add any application server as a member of the service integration bus. A messaging engine is created automatically on that bus for that application server.

   b. Create a WebSphere MQ client link for the application server and assign it to one messaging engine on the service integration bus. The MDB application connects to the JMS queues through the WebSphere MQ client link. The application server on which the MDB application is deployed does not have to be added to any service integration bus.

   c. For each resource of Version 5 embedded messaging, create an equivalent **Version 5 default messaging provider** resource.

      The administrative name for the embedded messaging JMS resources is changed from **WebSphere JMS Provider** resources to **V5 default messaging provider** resources. For example, in the administrative console the queue connection factory can be found by clicking **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queue connection factories**.

   d. For each JMS queue defined on the Version 5.1 application server, create a new bus queue with the same name as the Version 5.1 JMS queue, and create a message point assigned to the messaging engine. Messages sent to the JMS queues are stored and processed at the message point.

   The Version 5 embedded messaging JMS resources have been migrated to V5 default messaging JMS resources.

2. If any Version 5.1 default messaging JMS topic connection factory has the Port property set to DIRECT, **you must** change it to QUEUED before use with the default messaging provider. For example, after migrating the application server, use the administrative console in the later version to complete the following steps:
   a. Display the Version 5.1 default messaging JMS topic connection factory Click **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Topic connection factories** > ***factory_name***.
   b. For the **Port** field, select the QUEUED option.
   c. Click **OK**.
   d. Save your changes to the master configuration.

Results

After migrating the application server, the basic stand-alone application server scenario becomes as shown in the following figure Figure 3.

- The JMS application can continue to access the Version 5.1 JMS resources, which are now managed as Version 5.1 default messaging JMS resources implemented by the default messaging provider in the later version.
- The JMS application communicates with the Version 5.1 JMS resources through the WebSphere MQ client link and the messaging engine. This is invisible to the JMS application.
- The JMS resources, a JMS queue connection factory, shown as JMS QCF(V5), and a JMS queue, shown as JMS Q(V5), are managed as Version 5.1 default messaging JMS resources.
- The new bus queue, shown as Bus Q, is managed as a resource of the service integration bus. Messages for JMS Q(V5) are stored and processed by the message point for the associated bus destination, a queue point shown as BusQ@ME.
- The WebSphere MQ client link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by Version 5.1 JMS applications and the protocols used by the messaging engines in the later version of the product.

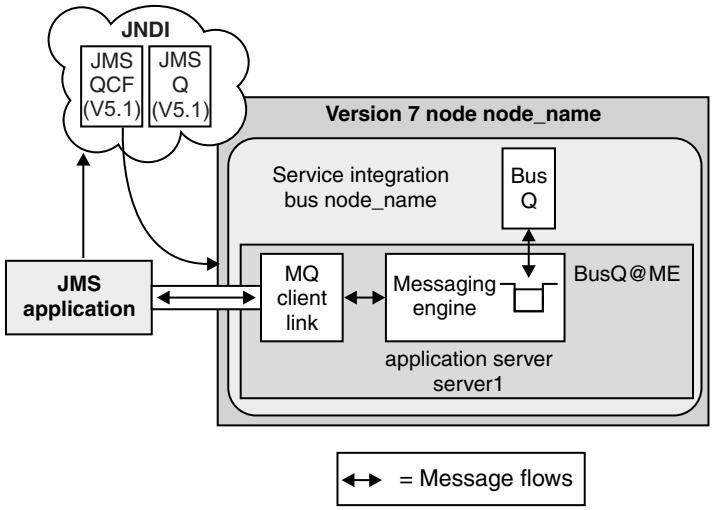


Figure 3. WebSphere Application Server Version 5 JMS application scenario after migration. This figure shows an example stand-alone application server scenario after migrating the application server to WebSphere Application Server Version 7.0. The JMS resources are now managed as Version 5 default messaging JMS resources implemented by the Version 7.0 default messaging provider. Also, a WebSphere MQ client link and bus queue have been created and assigned to the messaging engine, to enable JMS applications developed for WebSphere Application Server Version 5.1 to use the JMS resources.

What to do next

Security tip: If you have configured authorization level security on Version 5.1 it cannot be migrated to the later version. The migration tool cannot migrate authorization security for you and manual configuration is needed.

You should replace the Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources as soon as is conveniently possible (after all JMS applications that use those resources have been moved onto the later version of the product).

You should define any new JMS resources as resources in the later version; for example, as described in Configuring resources for the default messaging provider.

Migrating a network deployment configuration from Version 5 embedded messaging

Migrate a WebSphere Application Server Network Deployment environment from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in later versions.

Before you begin

Before migrating a WebSphere Application Server Version 5.1 node, you need to stop Version 5.1 JMS applications using the JMS queues that are to be migrated.

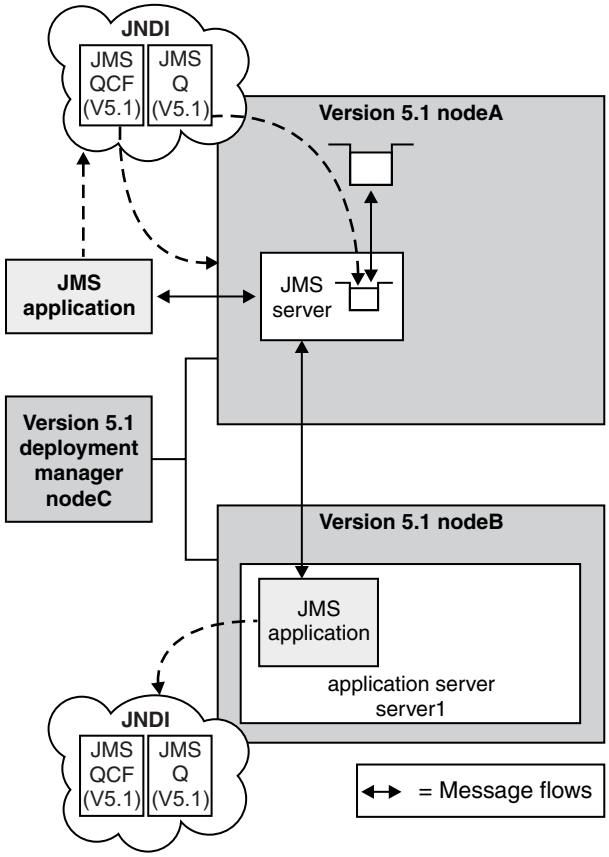
- Stop all message-producing JMS applications in the WebSphere Application Server Version 5.1 environment. For example, you can use the administrative console to stop the applications, as described in Starting or stopping enterprise applications.
- Allow all message-consuming JMS applications (including those consuming publications as a result of durable subscriptions) to continue until all the JMS queues are drained, then stop those applications.

About this task

When migrating a WebSphere Application Server Version 5.1 node to a later version, you do not need to make any changes to JMS applications; they can continue to use their same deployment and installation, and their same configurations of Version 5.1 JMS resources, apart from one exception which is described below. For a more detailed explanation, see “Migration from Version 5 embedded messaging” on page 26.

Consider the basic network deployment scenario before migration, as shown in the following figure Figure 4 on page 40.

- The JMS application uses JNDI to look up the JMS resources in the WebSphere Application Server namespace.
- The JMS resources, in this example a JMS queue connection factory (shown as JMS QCF) and a JMS queue (shown as JMS Q), can be configured as server resources or in the client container . The JMS resources are migrated without change except that if a topic connection factory has the Port property set to DIRECT, **you must** change it to QUEUED before use with the default messaging provider.
- The JMS queue connection factory creates connections to the JMS server on nodeA, as defined by the Node property of the connection factory. The connection factory can be configured to connect to a JMS server on any node in the deployment manager cell, and by default connects to the JMS server on the same node as the JMS application.
- WebSphere Application Server Version 5 embedded messaging uses WebSphere MQ technology, and is implemented through a JMS server that runs as a separate server on the node. The administrator has defined the name of the JMS queue, Q, to the JMS server. The JMS application uses WebSphere MQ client protocols to communicate with the JMS server.



This figure shows the example network deployment scenario before migrating any nodes to the later version. The JMS application is supported by an application server running on nodeB, and uses JMS resources configured to use a JMS server on nodeA. The cell is managed by the deployment manager on nodeC. The JMS application can be running within the application server or as a JMS client application.

Figure 4. WebSphere Application Server Network Deployment Version 5.1 JMS application scenario before migration.

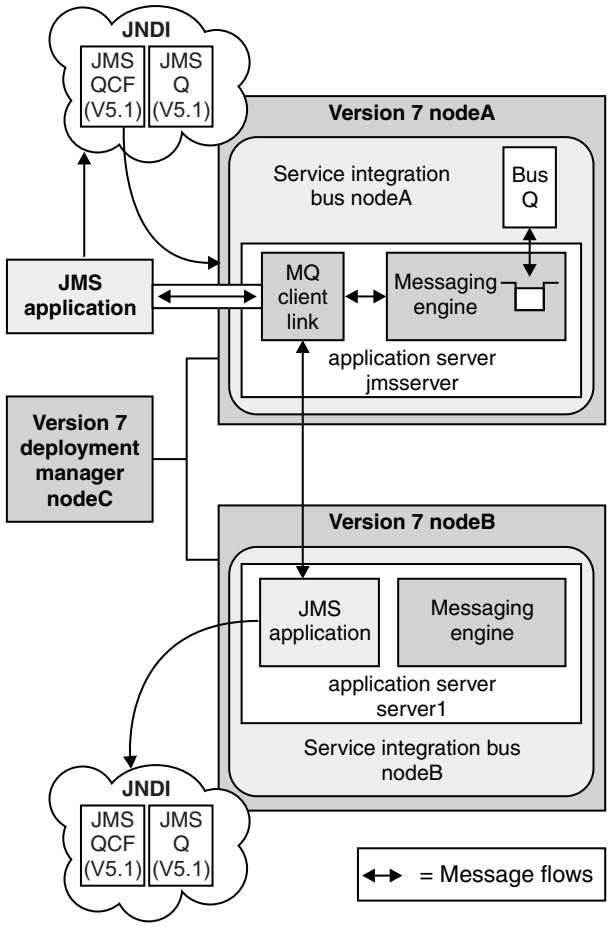
To migrate a WebSphere Application Server Network Deployment environment from Version 5 embedded messaging to the default messaging provider in a later version, complete the following steps:

Procedure

1. Migrate the WebSphere Application Server node to the later version. See the documentation about migrating product configurations. At this point, you have a cell in the later version, which is managed by a deployment manager at that version, with two Version 5 nodes (and their node agents).
2. Use the migration tools to migrate the Version 5.1 application server nodes to the later version.
3. If any Version 5.1 default messaging JMS topic connection factory has the Port property set to DIRECT, **you must** change it to QUEUED before use with the default messaging provider of the later version. For example, use the administrative console to complete the following steps:
   a. Display the Version 5.1 default messaging JMS topic connection factory Click **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Topic connection factories** > ***factory_name***.
   b. For the **Port** field, select the QUEUED option
   c. Click **OK**.
   d. Save your changes to the master configuration.

Results

After migrating all the nodes in the cell, the scenario then becomes as shown in the following figure Figure 5.



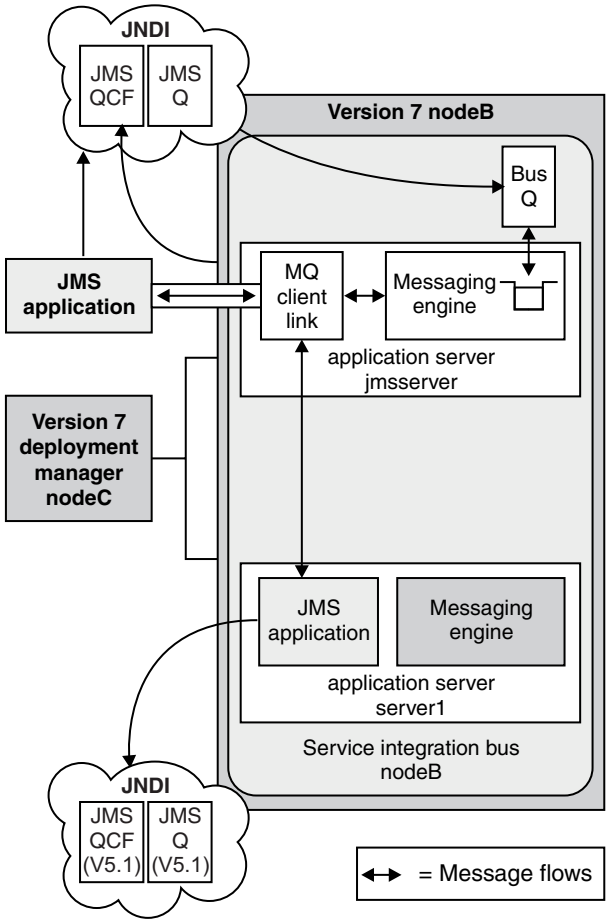
This figure shows the example network deployment scenario after migrating all the nodes to WebSphere Application Server Version 7.0, but before replacing JMS resources developed for WebSphere Application Server Version 5.1 with equivalent Version 7.0 JMS resources. The WebSphere MQ client link for the messaging engine on bus B enables JMS applications to use the Version 5.1 JMS resources implemented by the Version 7.0 default messaging provider.

Figure 5. WebSphere Application Server Network Deployment Version 5.1 JMS application scenario after migration stage 2.

The JMS application can continue to access the Version 5.1 JMS resources, which are now managed as Version 5.1 default messaging JMS resources implemented by the WebSphere Application Server Version 7.0 default messaging provider. You can use the Version 7.0 administrative console to manage the JMS resources as Version 5.1 default messaging JMS resources.

There are two variants on this migration:

- If the application server and JMS server were on the same host, the migration creates the same pair of service integration buses on the one host of the later version, as shown in the following figure Figure 6 on page 42.
- If JMS applications on a node of a later version have to use JMS resources provided by a JMS server on a Version 5.1 node in the cell, they can do so through the Version 5.1 default messaging JMS resource definitions.



This figure shows an example network deployment scenario after migrating the deployment manager nodeC and a server node that hosts both a JMS server and application server. The JMS server developed for WebSphere Application Server Version 5.1 has been recreated as a Version 7.0 application server with a messaging engine in its own service integration bus, shown as bus nodeB. Also, a WebSphere MQ client link has been created for the messaging engine on bus nodeB, to enable JMS applications developed for WebSphere Application Server Version 5.1 to use the JMS resources implemented by the Version 7.0 default messaging provider.

Figure 6. WebSphere Application Server Network Deployment Version 5.1 JMS application scenario after migration of a combined JMS server - application server node

What to do next

You should replace the Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources of the later version as soon as is conveniently possible (that is, after all the JMS applications that use those resources have been moved onto the later version).

You should define any new JMS resources as resources in the later version; for example, as described in Configuring resources for the default messaging provider.

Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 1

You can migrate a message-driven bean application from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in later versions.

Before you begin

Before migrating a WebSphere Application Server Version 5.1 node, you need to stop Version 5.1 JMS applications using the JMS queues that are to be migrated.

- Stop all message-producing JMS applications in the WebSphere Application Server Version 5.1 environment. For example, you can use the administrative console to stop the applications, as described in Starting or stopping enterprise applications.
- Allow all message-consuming JMS applications (including those consuming publications as a result of durable subscriptions) to continue until all the JMS queues are drained, then stop those applications.

About this task

This topic provides a contextual description of the migration, then a summary of the steps involved.

The migration of the MDB application is part of the migration of the Version 5.1[1] node, called wasA, on which it runs. When migrating the WebSphere Application Server Version 5.1 node to later versions, you do not need to make any changes to the MDB application; it can continue to use the same deployment and installation, and the same configurations of Version 5.1 JMS resources. However, to complete the migration, you should replace the listener port used by the MDB application with a JMS activation specification.

Consider the example scenario shown, before migration, in the following figure.

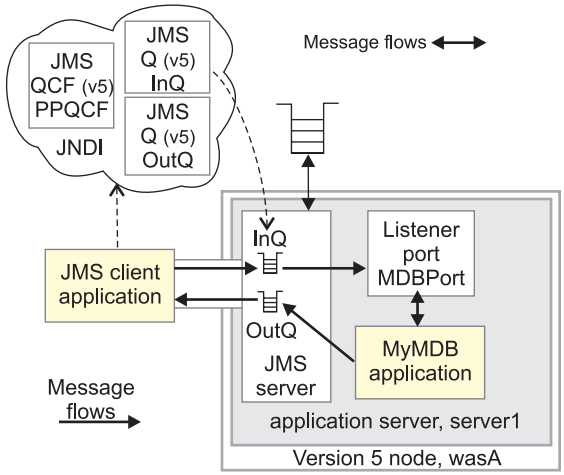


Figure 7. WebSphere Application Server Version 5.1 single-node MDB application scenario before migration

- The JMS resources are defined on WebSphere Application Server Version 5.1 as embedded messaging JMS resources:

  WebSphere Queue connection factory, PPQCF

Name:	PPQCF
JNDI Name:	jms/SamplePPQCF

  All other properties have default settings. By default, the connection factory creates connections to the JMS server on the same node.

1. To make reading easier in this topic, the abbreviation “Version 5.1” is sometimes used to refer to “WebSphere Application Server Version 5.1”. For example, “Version 5.1 JMS resources” refers to JMS resources provided by WebSphere Application Server Version 5.1.

WebSphere Queue, InQ

Name:	InQ
JNDI Name:	jms/SampleInputQueue

All other properties have default settings.

WebSphere Queue, OutQ

Name:	OutQ
JNDI Name:	jms/SampleOutputQueue

All other properties have default settings.

- The MDB application, MyMDB, is installed on the application server called server1.
- The listener port called MDBPort is defined on the application server and references the defined JMS queue connection factory and JMS input queue.

Name:	MDBPort
Initial state:	Started
Connection Factory JNDI Name:	jms/SamplePPQCF
Destination JNDI Name:	jms/SampleInputQueue

- The use of these resources in a message flow is:
  1. The JMS client application uses JNDI to look up the JMS resources in the WebSphere Application Server JNDI namespace. The client puts a message on the input queue.
  2. The message-driven bean in the MyMDB application uses the listener port to listen for messages arriving on the input queue. When a message is put on the input queue, the onMessage method of the message-driven bean is called, and the message-driven bean application puts a reply message on the output queue.
  3. The JMS client application gets the reply message from the output queue.
- WebSphere Application Server Version 5 embedded messaging uses WebSphere MQ technology, and is implemented through a JMS server that runs as a service within the application server. The JMS client application uses WebSphere MQ client protocols to communicate with the JMS server.

After migrating the node, the basic single-node scenario becomes as shown in the following figure, Figure 8 on page 46.

- The JMS application communicates with the JMS resources developed for WebSphere Application Server Version 5.1, through the WebSphere MQ client link and the messaging engine on the service integration bus. This is all invisible to the JMS application.
- The JMS resources, a JMS queue connection factory (shown as V5 JMS QCF) and a JMS queue (shown as V5 JMS Q), are managed as Version 5.1 default messaging JMS resources.

You should then replace the Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources in the later version as soon as is conveniently possible (for example, after any Version 5.1 JMS client applications have been migrated onto the later version).

To migrate the MDB application from Version 5 embedded messaging to the default messaging provider in the later version, complete the following steps:

Procedure

1. Migrate the WebSphere Application Server node to the later version. See the documentation about migrating product configurations.

As part of the task for migrating a node, you should complete the following steps to continue to use Version 5.1 default messaging JMS resources:

a. Create a service integration bus, and add an application server to that bus.

   The default messaging provider in later versions is based on one or more service integration buses. JMS destinations are assigned to a service integration bus. To make use of resources through service integration technologies, you can add any application server as a member of the service integration bus. A messaging engine is created automatically on that bus for that application server.

b. Create a WebSphere MQ client link for the node and assign it to one messaging engine on the service integration bus. The MDB application connects to the JMS queues through the WebSphere MQ client link. The application server on which the MDB application is deployed does not have to be added to any service integration bus.

c. For each resource of Version 5 embedded messaging, create an equivalent **Version 5 default messaging provider** resource.

   The administrative name for the embedded messaging JMS resources is changed from **WebSphere JMS Provider** resources to **V5 default messaging provider** resources. For example, in the administrative console the queue connection factory can be found by clicking **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queue connection factories**.

d. For each JMS queue defined on the Version 5.1 application server, create a new bus queue with the same name as the Version 5.1 JMS queue, and create a message point assigned to the messaging engine. Messages sent to the JMS queues are stored and processed at the message point.

The Version 5 embedded messaging JMS resources have been migrated to Version 5.1 default messaging JMS resources.

2. If any JMS application uses the Version 5 embedded messaging provider DIRECT port for publish/subscribe messaging, as set on the WebSphere topic connection factory, change the Port property of the connection factory to QUEUED before use with the default messaging provider of the later version.

Results

After migrating the Version 5.1 node, the MDB application scenario becomes as shown in the following figure:

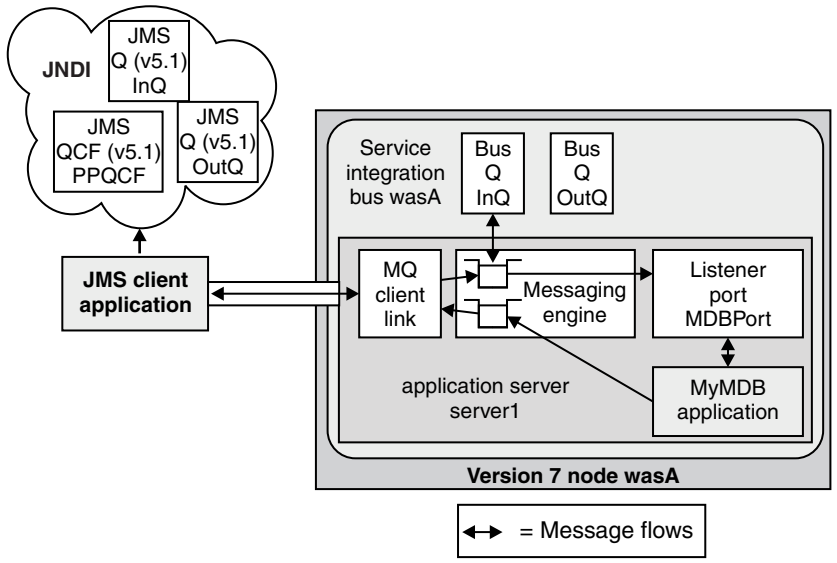


Figure 8. WebSphere Application Server Version 5.1 node after migration

The MDB application can continue to access the JMS resources, which are now implemented through the WebSphere Application Server default messaging provider in the later version. You can use the administrative console of that later version to manage the JMS resources as Version 5.1 default messaging JMS resources.

The MDB application can continue to receive messages through the listener port.

What to do next

After migrating a Version 5.1 MDB application, you should complete the following steps:

1. You should replace Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources of the later version as soon as is conveniently possible (after all JMS applications that use those resources have been moved onto the later version).
2. Change the configuration of the MDB application to use a JMS activation specification instead of the listener port.
3. Re-deploy or re-install (with the Deploy EJB option selected) the MDB application.

These steps enable you to benefit from the better performance of the default messaging provider, and to exploit the use of multiple messaging engines in a service integration bus, and other default messaging functions enabled by service integration technologies.

You can replace JMS resources manually, for example by using the WebSphere Application Server administrative console. Alternatively, you can replace the resources by writing some scripting that retrieves the Version 5.1 property values then creates JMS resources in the later version with values appropriate to that environment and your use of the Version 5.1 properties.

For an example of migrating the MDB application from Version 5.1 default messaging JMS resources and listener port to default messaging provider JMS resources in a later version, including JMS activation specification, see “Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2” on page 47.

Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2

By following this example, you can migrate a message-driven bean application on a Version 6 or later node from using Version 5.1 default messaging JMS resources and listener port to using default messaging provider JMS resources.

About this task

This topic provides a contextual description of the migration, then a summary of the steps involved.

This example follows on from the example described in “Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 1” on page 42.

Consider the example scenario, before replacing the Version 5.1 JMS resources with JMS resources of the later version, shown in the following figure Figure 9.

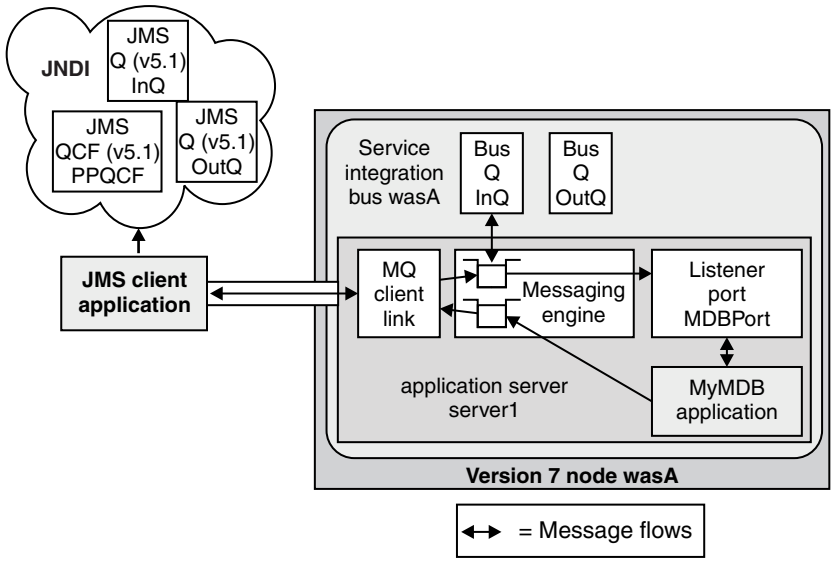


Figure 9. MDB application scenario before replacing the Version 5 JMS resources with JMS resources of a later version

- The JMS resources, a JMS queue connection factory (PPQCF) and JMS queues (InQ and OutQ), are managed as Version 5.1 default messaging JMS resources. The administrative name for the embedded messaging JMS resources is changed from **WebSphere JMS Provider** resources to **V5 default messaging provider** resources. For example, in the administrative console of the later version the queue connection factory can be found by clicking **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queue connection factories**.
- On the administrative console, display the listener port by clicking **Servers -> Server Types -> WebSphere application servers -> *server_name* -> [Communications] Messaging** > **Message Listener Service** > **[Additional Properties] Listener Ports** > ***port_name***, where ***server_name*** is **server1** and ***port_name*** is **MDBPort**.

To replace the Version 5.1 default messaging JMS resources with equivalent default messaging provider JMS resources of the later version, complete the following steps:

Procedure

1. Delete the Version 5.1 JMS resources:

a. Display the collection list of Version 5.1 JMS queue connection factories. Click **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queue connection factories**.
b. Select the check box next to the queue connection factory, PPQCF.
c. Click **Delete**
d. Click **OK**.
e. Display the collection list of Version 5.1 JMS queues. Click **Resources -> JMS -> JMS providers -> V5 default messaging provider -> [Additional Properties] Queues**.
f. Select the check box next to the queues, INQ and OUTQ.
g. Click **Delete**
h. Click **OK**
i. Save your changes to the master configuration.

2. Create a JMS queue connection factory in the later version, to replace the Version 5.1 JMS queue connection factory. (If you want to use a unified JMS connection factory instead of the domain-specific JMS queue connection factory, you must also rewrite the MDB application to use JMS 1.1 interfaces). For example, use the administrative console to complete the following steps:
   a. Display the collection list of JMS queue connection factories for the default messaging provider. Click **Resources -> JMS -> JMS providers -> Default messaging provider -> [Additional Properties] Queue connection factories**
   b. Click **New**
   c. On the New JMS queue connection factory page, set the following properties:

Name:	PPQCF
JNDI Name:	jms/SamplePPQCF
Bus name	wasA

   All other properties have default settings. The name and JNDI name properties match the same properties on the Version 5.1 JMS queue connection factory. The connection factory creates connections to the service integration bus called wasA.
   d. Click **OK**.
   e. Save your changes to the master configuration.
3. Create JMS queues in the later version to replace the Version 5.1 JMS queues For example, use the administrative console to complete the following steps for the input JMS queue:
   a. Display the collection list of JMS queues for the default messaging provider. Click **Resources -> JMS -> JMS providers -> Default messaging provider -> [Additional Properties] Queues**.
   b. Click **New**
   c. On the New JMS queue page, set the following properties for the JMS queue called InQ that is backed by the existing bus destination also called InQ:

Name:	InQ
JNDI Name:	jms/SampleInputQueue
Queue name:	InQ

   All other properties have default settings. The Queue name property specifies the name of the bus queue that is used to store and process messages for the JMS queue.
   d. Click **OK** This returns you to the collection list of JMS queues.
   e. Click **New**
   f. On the New JMS queue page, set the following properties for the JMS queue called OutQ that is backed by the existing bus destination also called OutQ:

Name:	OutQ
JNDI Name:	jms/SampleOutputQueue
Queue name:	OutQ

All other properties have default settings. The Queue name property specifies the name of the bus queue that is used to store and process messages for the JMS queue.

g. Click **OK**.

h. Save your changes to the master configuration.

4. Create a JMS activation specification to replace the Version 5.1 listener port. The JMS activation specification is used to deploy the MDB application as a J2EE Connector Architecture (JCA) 1.5-compliant resource, as a listener on the JMS queue InQ. For example, use the administrative console to complete the following steps:

   a. Display the collection list of JMS activation specifications for the default messaging provider. Click **Resources -> JMS -> JMS providers -> Default messaging provider -> [Additional Properties] Activation specifications**.

   b. Click **New**

   c. On the New JMS activation specification page, set the following properties:

Name:	PPAS
Name:	jms/SamplePPAS
Destination JNDI Name:	jms/SampleInputQueue

All other properties have default settings.

d. Click **OK** This returns you to the collection list of JMS activation specifications.

e. Save your changes to the master configuration.

5. Save your changes to the master configuration. After replacing the Version 5.1 JMS resources with equivalents in the later version, the MDB application scenario becomes as shown in the following figure:

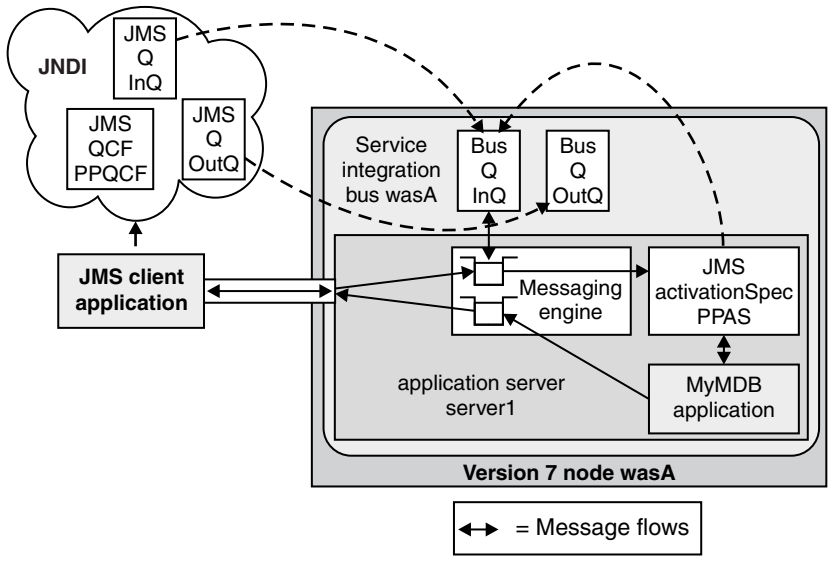


Figure 10. MDB application scenario after replacing the Version 5.1 JMS resources with Version 7.0 JMS resources

6. Redeploy the MDB application to use the JMS activation specification, as described in Deploying and administering enterprise applications. Ensure you select the **Do not overwrite existing bindings** option.

   Accept the defaults for all installation steps except for the following:

   - Delete the Listener Port binding
   - Set the activation specification binding to `jms/SamplePPAS`.

7. Click **OK**.
8. If no other Version 5.1 applications use the WebSphere MQ client link, delete it.
   a. Display the list of WebSphere MQ client links for the messaging engine. Click **Service integration -> Buses -> *bus_name* -> [Topology] Messaging engines -> *engine_name* -> [Additional Properties] WebSphere MQ client links**, where ***bus_name*** is **wasA** and ***engine_name*** is **wasA.server1-wasA**.
   b. Select the check box next to the link, Default.MQClientLink.
   c. Click **Delete**
   d. Click **OK**
9. Save your changes to the master configuration.

Results

You should now be able to use the MDB application with the JMS resources in the later version. On the Enterprise Applications panel (**Applications -> Application Types -> WebSphere enterprise applications**) ensure that the MDB application is started. There should be no errors displayed in the administrative console at this point. If there are any errors, check the SystemOut log for more information about the problem.

Using a Version 5.1 JMS client

There are certain limitations and restrictions to bear in mind when deciding whether to use a JMS client developed for WebSphere Application Server Version 5.1 with a later version of the product.

About this task

You can use a JMS client application developed for WebSphere Application Server Version 5.1 with a later version of the product without needing to recompile the application client. However, consider the following points:

Procedure

- A JMS client developed for WebSphere Application Server Version 5.1 can access JMS resources on a later version, which can be assigned to resources on a service integration bus, but cannot access directly default messaging JMS resources on that later version.

  Consider the JMS interoperation scenario, of a Version 5.1 JMS application that uses JMS resources provided by the default messaging provider on a node on a later version. The Version 5.1 JMS queue is backed by a bus queue, which is normal for a JMS queue on the later version, but there is no configured link between the Version 5.1 JMS queue and bus queue. The JMS application communicates with the bus queue through a WebSphere MQ client link and the messaging engine. To send messages to the bus queue or receive messages from the queue, the JMS application opens a connection on the WebSphere MQ client link. This is all invisible to the JMS application, but can be displayed and managed by the administrator.

- Application servers generate null responses for security flows sent by clients.

  Later versions of the product do not support the equivalent of channel exits. However, JMS clients developed for WebSphere Application Server Version 5.1 can invoke security, message send, and

message receive exits. The message send and receive exits are driven only at the client end, but clients can send. security flows to the application server. The application server generates null security responses for such security flows.

- Create a mapping between Version 5.1 user IDs and user IDs on the later version. Security checks are performed at connection time by using the system security context of the later version, and at the time of creating sessions against bus destinations. The user ID associated with the client is used. However, because user IDs for Version 5.1 JMS clients are 12 characters of type “char”, you have to create a mapping between these and the user IDs of the later version.
- Dynamic selectors are not supported. Later versions of the product do not provide support for dynamic selectors. That is, it is not possible to change the selector after having created a session. If you have to change a selector after having created a consumer session, the current consumer session is closed then a new consumer session is created with the new selector.
- Reconfigure DIRECT topic connections to QUEUED topic connections. Version 5.1 JMS clients that use a topic connection factory pointing to the high performance publish/subscribe provider must change their topic connection factory to point to the queue based publish/subscribe provider instead. To do this, change the Port property of the Topic Connection Factory from DIRECT to QUEUED.
- Application Server Facilities (ASF). The use of ASF is discouraged by Sun, with the statement that they are superseded by the Java EE Connector Architecture (JCA). ASF, used by JMS applications developed for WebSphere Application Server Version 5.1, is supported to enable connection to later versions. However, you should consider migrating your Version 5.1 JMS applications that use ASF to use JCA.
- Changes to writing and reading messages under syncpoint scope. A JMS application developed for WebSphere Application Server Version 5.1 uses the Version 5 embedded messaging provider to write messages under syncpoint scope to a queue, then read those messages before the writes have been committed. A JMS application that connects to the default messaging provider in later versions cannot do this, because messages are never visible to a consumer until the message producer commits the message.

Chapter 8. Migrating 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.

Migrating bootstrap addresses

WebSphere Application Server Versions 5 and 6.0.x support foreign cell bindings, but support is limited to a single bootstrap address and you cannot configure foreign cell bindings in the console. The single bootstrap address used in Versions 5.x and 6.0.x has been deprecated, replaced in Version 6.1 and later by a list of bootstrap addresses.

Before you begin

Configure server clusters for failover support as needed.

About this task

When a Version 5.x or 6.0.x cell is migrated to Version 6.1 or later, foreign cell binding configurations initially are left unchanged to maintain compatibility with existing wsadmin scripts. All nodes in the migrated cell, including those later than Version 6.0.x, continue to use the single bootstrap address setting to create bindings in the namespace.

This topic describes how to migrate any Version 5.x or 6.0.x bootstrap addresses to use Version 6.1 or later foreign cell bindings. Version 6.1 and later foreign cell bindings enable you to specify more than one bootstrap address and enhance failover support.

Procedure

1. Edit your Version 5.x or 6.0.x foreign cell binding scripts, replacing `bootstrapAddress` with `bootstrapAddresses`, the list property in Version 6.1 and later.

   The bootstrap address properties are in the context of the ForeignCell class. The fully qualified name for the Version 5.x or 6.0.x property now deprecated is the following:

   ```
   topology.cell:Cell/foreignCells/bootstrapAddress
   ```

   The fully qualified name for the bootstrap address list property in Version 6.1 and later is the following:

   ```
   topology.cell:Cell/foreignCells/bootstrapAddresses
   ```

   Version 6.1 and later foreign cell binding scripts should use the bootstrap address list property, `bootstrapAddresses`.
2. Run the convertScriptCompatibility command to convert each single bootstrap address configuration to a new bootstrap address list configuration containing that single bootstrap address.

© Copyright IBM Corp. 2011

Results

In the migrated cell, nodes still running Version 5 or 6.0.x will use one of the addresses in the bootstrap address list configuration, but no particular one, to create a foreign cell binding in the namespace.

Nodes running at a version later than Version 6.0.x ignore the old single bootstrap address setting in preference for the bootstrap address list. If the convertScriptCompatibility command encounters a foreign cell binding configuration with the old single bootstrap address and the newer bootstrap address list both set, the old single bootstrap address is removed and the bootstrap address list is left unchanged.

What to do next

Click **Environment** > **Naming** > **Foreign cell bindings** > ***foreign_cell_binding_name*** > **Bootstrap addresses** in the console and examine the collection on the Bootstrap addresses page to see if single bootstrap addresses migrated to the bootstrap address list.

Console foreign cell binding panels in Version 6.1 and later only display and update the bootstrap address list. They do not change the single bootstrap address configuration setting. If you use the console to edit a foreign cell binding configuration that existed when the cell was initially migrated, and you have not yet run the convertScriptCompatibility command, the updated binding configuration contains both the original single bootstrap address and whatever list of bootstrap addresses you configure in the console. In the migrated cell, nodes still running at Version 5.x or 6.0.x continue to use the single bootstrap address configuration setting to bind foreign cell bindings in the namespace.

Chapter 9. Migrating OSGi applications

This page provides a starting point for finding out how to migrate from the OSGi Applications Feature Pack in WebSphere Application Server Version 7, and how to use OSGi applications with different versions of the product.

trns: If you have a WebSphere Application Server Version 7 node that is augmented with the Feature Pack for OSGi Applications and JPA 2.0, and you federate the node into a WebSphere Application Server Version 8 cell, the addNode command does not succeed. This problem only occurs when you try to federate a Version 7 node that has already been augmented with the feature pack. The solution is to federate the Version 7 node before you augment it. For more information, see the troubleshooting tip An existing Version 7 application server augmented with the OSGi Applications feature cannot be federated into a Version 8 Deployment Manager.

trns:

- 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 subsequently 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 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.

© IBM Corporation 2010, 2011

Chapter 10. Migrating Scheduler service

This page provides a starting point for finding information about the scheduler service, a WebSphere programming extension responsible for starting actions at specific times or intervals.

Schedulers are persistent and transactional timer services that run Enterprise JavaBeans methods or send Java Message Service messages using any Java Message Service messages using any Java Platform, Enterprise Edition (Java EE) server application.

The scheduler service helps minimize IT costs and increase application speed and responsiveness by maximizing utilization of existing computing resources.

The scheduler service provides the ability to reliably process workloads using parallel processing and schedule resource-intensive tasks to process during low traffic off-hours.

Interoperating with schedulers

Interoperating with schedulers

Schedulers support forward compatibility. Tasks created in previous versions of WebSphere Application Server Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1 continue to run in WebSphere Application Server, Version 6.x schedulers. Tasks that you create using Version 6.x are not compatible with product schedulers from Version 5.x. Version 5.x schedulers do not run any Version 6.x tasks.

Schedulers and versions

All schedulers that are configured to use the same database and tables are considered a clustered scheduler. To guarantee that your tasks run correctly, all servers in a scheduler cluster must be at the same version. If the servers are at different versions, tasks created with a Version 6.x scheduler might not run. If a mixed-Version environment is required for a short period of time, then all scheduler poll daemons should be stopped on all Version 5.x servers to allow a Version 6.x server to run all tasks. This action allows the Version 6.x schedulers to obtain leases and run tasks that have been created with a Version 6.x scheduler.

Running tasks created with schedulers prior to Version 5.0.2 is not supported. See the topic, "Interoperating with the Scheduler service," in the WebSphere Application Server Enterprise Edition Version 5.0.2 information center for details on how to migrate these tasks to a more recent version. See the Information Center Library to access the Version 5.0.2 information center.

© Copyright IBM Corp. 2011

Chapter 11. Migrating 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.

Preserving a Version 5.1 gateway when migrating a cell

When you migrate a cell to a later version, any previously configured gateway on any Version 5.1 application server in the cell is replaced with an empty gateway. Therefore you must preserve your Version 5.1 gateway configurations before you migrate the cell.

Before you begin

WebSphere Application Server Version 5.0 is no longer supported, so you should migrate any existing gateways that are running in Version 5.0 application servers to run on application servers at the current level of the product.

Before you choose whether to preserve or migrate your Version 5.1 gateways, you should be aware of the context and restrictions that are described in Coexistence: Preserve or migrate a Version 5.1 gateway.

To migrate a gateway, see “Migrating a Version 5.1 Web services gateway configuration” on page 60.

About this task

Because a cell in a later version can contain application servers at different versions, you can continue to use Version 5.1 gateways that are running on Version 5.1 application servers even if you migrate the cell from Version 5.1 or Version 6 to Version 7.0 or later. However when you migrate a cell, any previously configured gateway on any Version 5.1 application server in the cell is replaced with an empty gateway. Therefore you must preserve your Version 5.1 gateway configurations before you migrate the cell.

To preserve Version 5.1 gateways and application servers when migrating a cell to a later version, complete the following steps. For information about how to save and restore the Version 5.1 gateway configurations, see the WebSphere Application Server Version 5.1 topic: Backing up and restoring a gateway configuration.

Procedure

1. Save the Version 5.1 gateway configurations.
2. Migrate the cell to the later version without also migrating the application servers, by referring to the documentation about migrating deployment managers.
3. Restore the Version 5.1 gateway configurations to the Version 5.1 application servers within the cell on the later version.

© IBM Corporation 2009

Migrating a Version 5.1 Web services gateway configuration

In WebSphere Application Server Version 5.1, the web services gateway was a separable component with its own user interface. In later versions of the product, the gateway is integrated into service integration bus-enabled web services, and re-implemented as a mechanism for extending and linking inbound and outbound services. You use a wsadmin command script to migrate an existing gateway configuration from a Version 5.1 application server to an application server or cluster on a later version.

Before you begin

Consider whether you have to migrate your existing gateways:

- WebSphere Application Server Version 5.0 is no longer supported, so you should migrate any existing gateways that are running in Version 5.0 application servers to run on application servers at the current level of the product.
- Web services gateways running on WebSphere Application Server Version 5.1 can, subject to certain restrictions, coexist with gateway instances running on Version 7.0 or later application servers.
- A Version 7.0 or later cell can contain Version 5.1, Version 6 and Version 7.0 or later application servers.

For more information, see Coexistence: Preserve or migrate a Version 5.1 gateway.

You can migrate a Version 5.1 gateway that is in production use without stopping the gateway; requester applications can then switch over to using the new gateway configuration while the existing Version 5.1 gateway continues to run.

About this task

The migration process takes a Version 5.1 gateway application whose configuration has been exported to an XML file and uses the exported XML file to configure the same gateway functions on a single application server or cluster on the later version. To do this you export the Version 5.1 gateway configuration, then run a script to migrate the exported configuration into a new gateway instance in an existing application server or cluster on the later version.

The Version 5.1 configuration is migrated as follows:

- As part of the migration process, a gateway instance is created automatically.
- Gateway services, target services and UDDI references are migrated directly.
- The definitions within the gateway of JAX-RPC handlers and handler lists are also migrated. You must ensure that the underlying handler classes are available at run time.
- Assignments of gateway services to specific channels are replaced by equivalent assignments to specific inbound port and endpoint listener pairs (because in later versions the functions of a channel are shared between an endpoint listener and an inbound port). Any use of an Apache SOAP channel is migrated to a SOAP over HTTP endpoint listener and inbound port.
- Existing filters are not migrated. The use of filters was deprecated in Version 5.1.1 and support for filters was removed in Version 7.0. The role formerly played by filters is now undertaken by a combination of JAX-RPC handlers and service integration bus mediations.
- Web service clients that are generated from the WSDL for the target service, rather than the gateway service, are flagged by default in later versions as an error. After migration, use the following troubleshooting tip to enable this approach to work in later versions: You choose to generate a web service client from the WSDL for the target service, rather than the gateway service. When you try to access the target service through the gateway, your client receives the following error message: “CWSIF0304E: The message body did not match any of the definitions in the WSDL”.
- If you used the Version 5.1 gateway service WSDL to generate your web service clients, and your WSDL binding and encoding style is not document literal, then after migration to a later version you must regenerate the client stubs by using the new gateway service WSDL. For more information, see

the following troubleshooting tip: You migrate a gateway from WebSphere Application Server Version 5.1 to a later version. When you try to access a gateway service, your client receives one or more error messages stating “No response body is available”, or “Null response message”, or “The message body did not match any of the definitions in the WSDL”.

- WS-Security bindings are migrated as bindings that comply with the WS-Security Draft 13 specification. However:
  - The final version (1.0) of the WS-Security specification (implemented in WebSphere Application Server Version 6) is not compatible with the Draft 13 version, and therefore use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6. Use of WS-Security Draft 13 is deprecated, and you should only use it to allow continued use of an existing web services client application that has been written to the WS-Security Draft 13 specification.
  - The WS-Security binding objects are only migrated if the migration process is run on the machine on which the target server is running in the case of a stand-alone server, or on the machine on which the deployment manager is running in a network deployment configuration.
  - Only WS-Security binding objects that are used by a Gateway Service or Target Service WS-Security configuration are migrated. Any binding objects that you create but do not use are not migrated. For example: If you have a WS-Security configuration that references a Signing Information object, and the Signing Information object references a Trust Anchor, then the Signing Information object and the Trust Anchor object are both migrated along with the WS-Security configuration that references them.

Note:

- The migration assumes that the external web addresses for the migrated services are unchanged. This assumption is based on the expectation that these addresses are associated with a web server rather than with the machine on which the gateway is hosted, and that the host name and port number for these addresses are therefore not affected. If in your configuration the external web addresses point to the gateway machine, modify the endpoint listener configuration after the migration process has completed.
- You can use WebSphere Application Server Network Deployment to migrate to a single server running under either configuration profile (stand-alone server or deployment manager). However, it is recommended that you migrate to a single server running under a deployment manager profile. If you migrate to a stand-alone server profile you cannot use the administrative console to subsequently modify your gateway configuration. For more information, see the troubleshooting tip The gateway panels in the administrative console are only available in WebSphere Application Server Network Deployment if you are working with a deployment manager profile.
- Service integration bus-enabled web services validate web service messages more thoroughly than is done in WebSphere Application Server Version 5.1. As a result, some client applications that use poorly-formed requests or responses (where the message parts are misnamed) and that work when using Version 5.1 are now identified as poorly-formed. For the steps to take to resolve the problem, see Toleration of poorly-formed SOAP messages.

To migrate an existing gateway configuration from a Version 5.1 application server to the gateway capability on an application server or cluster on a later version, complete the following steps:

Procedure

1. Remove any filters from your Version 5.1 gateway.

   You can migrate a gateway that contains filters. However filters do not work in later versions, so you might prefer to remove them from the configuration before migration by completing the following steps:

   a. Check whether your Version 5.1 gateway uses filters. For more information, see the WebSphere Application Server Version 5.1 topic: Listing and managing gateway-deployed filters.

   b. Remove any filters. For more information, see the WebSphere Application Server Version 5.1 topic: Removing filters from the web services gateway.

After migration, you can recreate your filter functions by using a combination of JAX-RPC handlers and service integration bus mediations. If you migrate a web services gateway that includes a routing filter, you can recreate the filter functions as described in Choosing a target service and port through a routing mediation.

2. Choose a target server or cluster that is a single server or cluster on the later version, and is part of a network deployment cell.
3. Configure the target server or cluster as a member of a service integration bus. For more information, see Configuring the members of a bus.
4. Configure a Service Data Objects (SDO) repository at cell scope for the target server or cluster.
5. If you are migrating any EJB bindings, and you want them to continue to use an RPC-encoded binding or any binding other than document literal, add a binding of the correct type to the EJB binding WSDL. This step is necessary because the Version 5.1 gateway default binding is *RPC-encoded*, whereas in later versions the default binding is *document literal*.
6. Ensure that the source (Version 5.1) application server is running, then use the Version 5.1 gateway user interface to back up the gateway configuration from the Version 5.1 application server as a *private* configuration. For more information, see the WebSphere Application Server Version 5.1 topic: Backing up a gateway configuration.
7. Optional: Stop the Version 5.1 application server.

   Note: If you are migrating a gateway that is in production use, keep the Version 5.1 gateway running until the gateway configuration on the later version is complete, then switch the requester applications over to using the new gateway configuration while the existing Version 5.1 gateway continues to run. However both versions of the gateway do not have to be running at the same time, and you might have to stop the Version 5.1 server before you start the server or cluster on the later version (for example if you are installing the server or cluster on the later version as a direct replacement for the Version 5.1 server, on the same machine and using the same port numbers).
8. Start the target application server or cluster on the later version and, for a single server or cluster within a managed cell, the deployment manager for the target cell.
9. Check that all the WSDL documents that were used to define the target services on the Version 5.1 application server are available at their given locations. If the WSDL location is a UDDI reference, check that the referenced UDDI registry is available.
10. Optional: If the gateway being migrated uses JAX-RPC handlers and handler lists, ensure that the underlying handler classes are available at run time.
11. To migrate the exported configuration into a new gateway instance in the application server or cluster on the later version, complete the following steps:
    a. Open a command prompt, then change to the *app_server_root*/util directory.
    b. Run the following command:

```
migratewsgw.ext -C=cell_name [-S=server_name -N=node_name]
                    [-X=cluster_name] -B=bus_name
                     -G=v5_gateway_configuration_file_name
                    [-H=administration_hostname] [-A=administration_port]
                    [-U=gateway_instance_name] [-P=object_prefix]
                    [-username=WAS_user_ID -password=WAS_password]
```

    where:

    - *.ext* is the file extension .bat for a Windows system, or .sh for a Unix or Linux system.
    - Square brackets ("[ ]") indicate that a parameter or set of parameters is optional in some circumstances.
    - Either *server_name* and *node_name* together (for a single server), or *cluster_name* (for a cluster), defines the server or cluster to which the gateway configuration is migrated.
    - *cell_name*, *server_name* and *node_name* (or *cluster_name*), *administration_hostname* and *administration_port* together define the connection to the application server (or cluster) on the

later version. *server_name* or *cluster_name* specifies the name of the target application server or cluster at which endpoint listeners and outbound port destinations are created. If you are migrating to a server or cluster that is part of a managed cell, then *administration_hostname* and *administration_port* define the host name and the SOAP administration port number of the deployment manager. If you are migrating to a server that is not part of a managed cell, then *administration_hostname* and *administration_port* define the host name and port number of the stand-alone server, and are optional. If they are omitted, the command assumes that the intended values are `localhost:8880` (that is, the WebSphere Application Server default values for a stand-alone server).

- *v5_gateway_configuration_file_name* is the full path and file name for the exported Version 5.1 private gateway XML configuration file.
- *bus_name* and *gateway_instance_name* together define the gateway instance that you are creating within this bus. The *gateway_instance_name* is only required if you want to create more than one gateway instance within this bus. If you omit this optional parameter, then a default name is assigned.
- *object_prefix* is a string used to prefix the names of the objects defined by the migration process. If omitted, the namespace URI (default value `urn:ibmwsgw`) for the migrated services is used instead.
- *WAS_user_ID* and *WAS_password* are required if the target application server or cluster is password-protected.

12. Optional: If the external web addresses for the migrated services are changed by the migration process, modify the endpoint listener configuration to update these addresses. You must do this if the external web addresses point to the gateway machine rather than to a web server, and you have migrated the gateway to a different machine or to a different port on the same machine.

What to do next

Note:

- If your Version 5.1 gateway used filters, recreate your filter functions by using a combination of JAX-RPC handlers and service integration bus mediations.
- If the gateway configuration includes any gateway services that have multiple target services, the Version 5.1 configuration might have used a routing filter to choose a particular target service. If this is the case, then you must further configure your migrated gateway to choose a target service and port through a routing mediation.
- A web services gateway on a later version uses more memory to process a message, so if you pass a large attachment through the migrated gateway you might get an out-of-memory error in the Java virtual machine. To solve this problem, increase the JVM heap size as described in Tuning bus-enabled web services.

Adding unique names to the bus authorization policy

How to update the authorization policy for the service integration bus with unique name entries.

About this task

You should carry out this task if you are migrating from WebSphere Application Server Version 6 to WebSphere Application Server Version 7.0 or later. In this task, you manually run the populateUniqueNames command to query the user repository for a selected bus for unique names, and add them to the authorization policy. If you do not manually run this command, the messaging engine performs the query, and adds the missing unique names to the authorizations policy, which adversely affects the start up time.

When you migrate from a Version 6 node to a Version 7.0 or later node, the authorization policy only contains the user and group security names; it does not contain the names in the user registry that

uniquely define each user and group. If an LDAP user registry is in use, the unique name is the distinguished name (DN). By default, only missing unique names are added to the authorization policy. If you set the **-force** parameter, all unique name entries added to the authorization policy

Procedure

1. Run a scripting command. For more information, see Starting the wsadmin scripting client using wsadmin scripting.
2. At the wsadmin command prompt, type the populateUniquenames command. The following example syntax queries the user repository for the unique names that match the security names for a bus called Bus 1, and adds the missing unique names to the authorization policy .

   ```
   AdminTask.populateUniquenames('[-bus Bus1]')
   ```
3. Save your changes to the master configuration repository. The following example presents the syntax:

   ```
   AdminConfig.save()
   ```

Results

The authorization policy for the bus is updated with the missing unique names.

Example

The following example updates all the unique name entries in the authorization policy for a bus called Bus 1.

```
AdminTask.populateUniqueNames(AdminTask.populateUniquenames('[-bus Bus1 -force TRUE]')
```

What to do next

Use the administrative console to administer bus security authorizations.

Migrating a messaging engine based on a data store

Depending on your existing settings, when migrating a messaging engine from older versions of WebSphere Application Server to Version 7.0 or later, you might have to create a new data store table.

About this task

When migrating from older versions of WebSphere Application Server to Version 7.0 or later, if you do not have the "create tables automatically" option selected, it will be necessary to create a new table (SIBOWNERO) in each of your messaging engine database schemas. DDL for the creation of this table can be generated by using the sibDDLGenerator tool located in the bin directory of your WebSphere Application Server installation.

Procedure

1. Use the sibDDLGenerator tool to generate the DDL for the creation of this table as described in Generating the DDL statements needed to create data store tables manually.
2. Send the output file to your database administrator to process.

Chapter 12. Migrating 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...

Interoperating transactionally between application servers

You can configure application servers so that transaction messages are sent and received between application servers at different versions of WebSphere Application Server. Depending on the version of the application server, you can set system properties, or use the transaction coordination authorization setting.

About this task

The transaction manager in WebSphere Application Server supports transactional interoperation with other transaction managers through either the CORBA Object Transaction Service (OTS) protocol, or, for JSR-109 compliant requests, the Web Services Atomic Transaction (WS-AT) protocol. Also, the transaction manager can coordinate XA resource managers and be coordinated by Java EE Connector Architecture 1.5 resource adapters.

Procedure

When administrative security is enabled for application servers at WebSphere Application Server Version 6.0.2 or later, you must disable transaction coordination authorization for such servers in the following situations:

- The server interoperates transactionally with application servers at a version earlier than WebSphere Application Server Version 6.0.2.
- The server interoperates transactionally with non-WebSphere Application Server servers.
- The server interoperates transactionally with other servers and the server is not in a Common Criteria EAL4 evaluated configuration.

  When administrative security is enabled, the transaction manager is configured by default for use in the Common Criteria EAL4 evaluated configuration.

The transaction coordination authorization setting controls only the transaction protocol messages between servers that are used to coordinate the completion of a transaction. It does not affect application messages or the security of the server. When transaction coordination authorization is enabled, the server verifies that the sending server is authorized to handle prepare, commit, rollback, and one-phase commit messages.
To disable transaction coordination authorization on a server, use the following steps.

1. In the administrative console, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **[Container Settings] Container Services** > **Transaction Service**.
2. Clear the **Enable transaction coordination authorization** check box.
3. Click **Apply** or **OK**.
4. Save your changes to the master configuration.
5. Restart the server.

© Copyright IBM Corp. 2011

Chapter 13. Migrating 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. More introduction...

Migrating web application components

Migrating web application components from WebSphere Application Server Version 5.x

Migration of web applications deployed in previous versions of WebSphere Application Server is usually not necessary. Versions 2.2 and later of the Java Servlet specification and versions 1.2 and 1.4 of the JavaServer Pages (JSP) specifications are still supported unless the behavior was changed in the Servlet 3.0 or JSP 2.1 specifications. These changes are generally available in more detail in their corresponding specification.

About this task

Servlet migration might be a concern if your application:

- Implements a WebSphere Application Server internal servlet to bypass a WebSphere Application Server Version 4.x single application path restriction
- Extends a PageListServlet that relies on configuration information in the servlet configuration XML file
- Calls the `response.sendRedirect` method for a servlet using the `encodeRedirectURL` function or starting within a non-context root
- Depends on a default Content-Type response header being set or the behavior of a setContentType call after a getWriter call is made. The behavior is set by WebSphere Application Server version level using the web container custom property com.ibm.ws.webcontainer.contenttypecompatibility with a value of V4, V5, V6. or V7. The behavior for each version is described below.

Table 5. Web container custom properties.. Describes the version behavior for each version.

	Version 4	Version 5	Version 6	Version 7
Default Content-Type	text/html	text/html; charset=<default_encoding>	none	none
Append Charset on getWriter if the property does not exist on Content-Type Example: response.setCharacterEncoding("UTF-8"); response.setContentType("text/xml"); response.getWriter();	text/html	text/html	text/xml; charset=UTF-8	text/xml; charset=UTF-8
Remove charset from the Content-Type property if the setContentType property is called after getWriter with a ";charset=" portion Example: setContentType("text/html;charset=ISO-8859-7"); getWriter(); setContentType("text/xml;charset=UTF-8");	text/html	text/html	text/html	text/xml; charset=ISO-8859-7

JSP migration might be a concern if your application references JSP page implementation classes in unnamed packages, or if you install WebSphere Application Server Version 4.0.1 EAR files (deployed in Version 4.0.1 with the JSP `Precompile` option), in Version 5.x. You need to recompile all JSP pages when migrating from WebSphere Application Server Version 5.x.

© Copyright IBM Corp. 2011

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.

Follow these steps if migration issues apply to your web application:

Procedure

1. IBM internal servlets are used to enable special behavior such as file serving and serving servlets by class name. If a migrated application references internal servlets, ethe best practice is to enable or disable the functionality through the IBM WebSphere extensions XMI file, `ibm-web-ext.xmi`, located in each web module WEB-INF directory or by using an assembly tool.
2. If using these configuration options are not viable, then verify that the package names for the following internal servlets match what is used in your version 7 web deployment descriptor.

Feature	Configuration Option	Servlet Class
Directory browsing	directoryBrowsingEnabled="true"	com.ibm.ws.webcontainer.servlet.DirectoryBrowsingServlet
Auto mapping of servlet paths	serveServletsByClassnameEnabled="true:	com.ibm.ws.webcontainer.servlet.SimpleFileServlet
File serving	fileServingEnabled="true"	com.ibm.ws.webcontainer.servlet.FilterProxyServlet

3. Migrate servlets that extend PageListServlet and rely on configuration information in the associated XML servlet configuration file. In Version 4.0.1, the XML servlet configuration file provides configuration data for page lists and augments servlet configuration information. This file is named as either *servlet_class_name*.servlet or *servlet_name*.servlet, and is stored in the same directory as the servlet class file.
4. Add the web container custom property, com.ibm.ws.webcontainer.contenttypecompatibility, with a value of V4, V5, V6, V7, and so on. The value is determined by the version that the application is dependant on. Because this property is a global setting, you must consider the effect on other applications.

JavaServer Pages migration best practices and considerations

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

Classes from the unnamed or default package

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

Page encoding for JSP documents

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

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

As of JSP 2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the pageEncoding attribute of those pages is only checked to make sure that it is consistent with the page encoding determined as per the XML specification. As a result of this change, JSP documents that rely on their page encoding to be determined from their pageEncoding attribute are no longer decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

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

`web.xml` file version

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

1. EL expressions are ignored by default in JSP 1.2 applications. When you upgrade a web application to JSP 2.0, EL expressions are interpreted by default. You can use the escape sequence, `\$`, to escape

EL expressions that should not be interpreted by the container. Alternatively, you can use the isELIgnored page directive attribute, or the `<el-ignored>` configuration element to deactivate EL for entire translation units. Users of JSTL 1.0 must upgrade their taglib imports to the JSTL 1.1 URI or use the **_rt** versions of the tags, for example, use **c_rt** instead of **c**, or **fmt_rt** instead of **fmt**.

2. Web applications that contain files with an extension of `.jspx` will have those files interpreted as JSP documents, by default. You can use the JSP configuration element <is-xml> to treat `.jspx` files as regular JSP pages, but there is no way to disassociate `.jspx` from the JSP container.
3. The escape sequence \$ was not reserved in JSP 1.2. The output for any template text or attribute value that displays as \$ in JSP 1.2 was \$, however, the output now is just $.

jsp:useBean tag

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

Generated packages for JSP classes

Any reliance on generated packages for JSP classes results in non-portable JSP files. Packages for generated classes are implementation-specific and therefore do not rely on these packages.

JspServlet class

JspServlet classes are no longer used. Any reliance on the existence of a JspServlet class causes unrecoverable error problems.

Important: While the JSP classes migrate successfully, if you are precompiling JSP classes, precompile them again on the target server level. Annotation processing is supported as of JSP 2.1. Recompile the JSP classes to take advantage of this processing.

JavaServer Faces migration

In WebSphere Application Server V8.0, the default JavaServer Faces (JSF) implementation has changed to MyFaces.

Note: The default JavaServer Faces implementation is now MyFaces 2.0, which provides full compliance with the updated JSF 2.0 specification.

Choosing the correct JSF implementation

After updating WebSphere Application Server, the initial state for all applications will be to use the MyFaces 2.0 JSF implementation. The Sun Reference Implementation (RI) is still optionally configurable for migrated applications that require behaviors specific to this implementation.

Note: The RI is included in a deprecated state and only at the JSF 1.2 specification level.
Some examples when an application might require the RI include, but are not limited to:

- Applications or dependant libraries that directly extend RI classes instead of JSF API classes.
- Applications that use context parameters that are specific to the RI and do not have an equivalent context parameter in MyFaces.

IBM JavaServer Faces widget library

You must update widget library to version 3.1.6 or higher to enable compatibility with the changes in the JSF 2.0 specification. Obtain a compatible version of JWL by upgrading IBM Rational® Application Developer for WebSphere to 7.5.5.2 or later or installing IBM Rational Application Developer for WebSphere V8.0.

Attention: JWL is deprecated and does not work with facelets-based JSF pages; it only works with JSF pages that are built using the JavaServer Pages (JSP) technology.

JSF 2.0 exception handling

The default behavior for JSF when unexpected exceptions occur during JSF lifecycle processing has changed. Prior to JSF 2.0, unexpected exceptions during lifecycle processing were hidden by the runtime environment. This behavior is no longer the case with JSF 2.0. Exceptions are now published to the new ExceptionHandler API, as described in section 6.2 of the JSF 2.0 specification.

Add the following code snippet in the `<factory>` section of the faces-config.xml file to any applications that have a requirement on the previously-defined behavior:

```
<exception-handlerfactory>
  javax.faces.webapp.PreJsf2ExceptionHandlerFactory
</exception-handlerfactory>
```

Migration scenario for the getHeaderNames method

WebSphere Application Server Version 8 supports the Java Servlet 3.0 API.

As part of the Java Servlet 3.0 API, the following method is included in the HttpServletResponse:

```
Collection<String> getHeaderNames()
```

The com.ibm.websphere.servlet.response.StoredResponse class that existed before WebSphere Application Server Version 8 had the following method:

```
Enumeration getHeaderNames()
```

Because StoredResponse implements HttpServletResponse, StoredResponse no longer compiles without changing the return type of its getHeaderNames method to a type implementing Collection <String>. This change breaks any existing application using the old Enumeration return type.

Note: 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.

JavaServer Pages migration

JavaServer Pages migration best practices and considerations

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

Classes from the unnamed or default package

As of JSP 2.0, referring to any classes from the unnamed or default package is not allowed. This can result in a translation error on some containers, specifically those that run in a JDK 1.4 or greater environment which also breaks compatibility with some older JSP applications. However, as of JDK 1.4,

importing classes from the unnamed package is not valid. Refer to the Java 2 Platform Compatibility with Previous Releases web site for details. Therefore, for forwards compatibility, applications must not rely on the unnamed package. This restriction also applies for all other cases where classes are referenced, such as when specifying the class name for a tag in a Tag Library Descriptor (TLD) file.

Page encoding for JSP documents

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

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

As of JSP 2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the pageEncoding attribute of those pages is only checked to make sure that it is consistent with the page encoding determined as per the XML specification. As a result of this change, JSP documents that rely on their page encoding to be determined from their pageEncoding attribute are no longer decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

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

`web.xml` file version

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

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

jsp:useBean tag

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

Generated packages for JSP classes

Any reliance on generated packages for JSP classes results in non-portable JSP files. Packages for generated classes are implementation-specific and therefore do not rely on these packages.

JspServlet class

JspServlet classes are no longer used. Any reliance on the existence of a JspServlet class causes unrecoverable error problems.

Important: While the JSP classes migrate successfully, if you are precompiling JSP classes, precompile them again on the target server level. Annotation processing is supported as of JSP 2.1. Recompile the JSP classes to take advantage of this processing.

JavaServer Pages migration

JavaServer Pages migration best practices and considerations

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

Classes from the unnamed or default package

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

Page encoding for JSP documents

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

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

As of JSP 2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the pageEncoding attribute of those pages is only checked to make sure that it is consistent with the page encoding determined as per the XML specification. As a result of this change, JSP documents that rely on their page encoding to be determined from their pageEncoding attribute are no longer decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

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

web.xml file version

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

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

jsp:useBean tag

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

Generated packages for JSP classes

Any reliance on generated packages for JSP classes results in non-portable JSP files. Packages for generated classes are implementation-specific and therefore do not rely on these packages.

JspServlet class

JspServlet classes are no longer used. Any reliance on the existence of a JspServlet class causes unrecoverable error problems.

Important: While the JSP classes migrate successfully, if you are precompiling JSP classes, precompile them again on the target server level. Annotation processing is supported as of JSP 2.1. Recompile the JSP classes to take advantage of this processing.

HTTP session migration

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

Migration from Version 5.x

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

Chapter 14. Migrating 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.

Migrating web services

Web services migration scenarios: JAX-RPC to JAX-WS and JAXB

This topic explains scenarios for migrating your Java API for XML-based RPC (JAX-RPC) web services to Java API for XML-Based Web Services (JAX-WS) and Java Architecture for XML Binding (JAXB) Web services.

Changes in the tooling can be largely hidden from the user by a development environment like the assembly tools available with WebSphere Application Server. Also, depending on the data specified in the XML, some methods that were used for the JAX-RPC service can be used with little or no change in a JAX-WS service. There are also conditions that do require changes to be made in the JAX-WS service.

Because Java EE environments emphasize compatibility, most application servers that offer support for the newer JAX-WS and JAXB specifications continue to support the previous JAX-RPC specification. A consequence of this is that existing web services are likely to remain JAX-RPC based while new web services are developed using the JAX-WS and JAXB programming models.

However, as time passes and applications are revised and rewritten, there might be times when the best strategy is to migrate a JAX-RPC based web service to one that is based on the JAX-WS and JAXB programming models. This might result from a vendor choosing to provide enhancements to qualities of service that are only available in the new programming models. For example, SOAP 1.2 and SOAP Message Transmission Optimization Mechanism (MTOM) support are only available within the JAX-WS 2.x and JAXB 2.x programming models and not JAX-RPC.

The following information includes issues that you might have while migrating from JAX-RPC to JAX-WS and JAXB.

Attention: When the terms *migrate*, *migrating*, and *migration* are used in this topic, unless there is some statement to indicate otherwise, the move from a JAX-RPC to a JAX-WS and JAXB environment is being described.

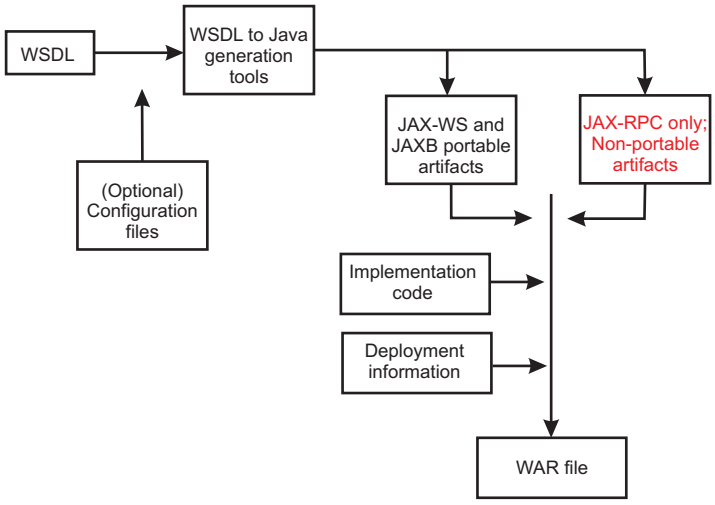
Comparison of JAX-RPC to JAX-WS and JAXB programming models

If you are looking for additional information that describes the changes between the JAX-RPC and JAX-WS and JAXB programming models, IBM developerWorks has published several web services hints and tips topics.

© Copyright IBM Corp. 2011

Development models

The high level combined development models for JAX-RPC and JAX-WS and JAXB are shown the following image:



At a high level, the models are very similar. The only difference is that the JAX-RPC model produces both portable and non-portable artifacts; the JAX-WS and JAXB model does not produce non-portable artifacts.

The image displays the following components:

- **Web Services Description Language (WSDL) file:** A document conforming to the WSDL recommendation as published by the W3C organization.
- **(Optional) Configuration files**
- **WSDL to Java generation tool:** The specifications describe the mappings required, not the name of the tools. Typically, the tool for JAX-WS and JAXB is wsimport, and for JAX-RPC the tools are wscompile and wsdeploy.
- **Portable artifacts:** These are files generated by the WSDL to Java generation tool that have well-defined mappings in the JAX-RPC or JAX-WS and JAXB, whichever is applicable, specifications. These artifacts can be used without modification between different implementations of JAX-RPC or JAX-WS and JAXB.
- **Non-portable artifacts:** These are files generated by the WSDL to Java generation tool that do not have well-defined mappings in the JAX-RPC or JAX-WS and JAXB, whichever is applicable, specifications. These artifacts generally require modification between different implementations of JAX-RPC or JAX-WS/JAXB.
- **Implementation code:** This is the code that you need to meet particular requirements for implementation.
- **Deployment information:** Additional information that is needed to run the application in a particular environment.
- **Web archive (WAR) file:** In the context of the SampleService image, a WAR file is an archive file that contains all items needed to deploy a web service into a particular environment. This is sometimes called a *cooked WAR file*.

The Development and runtime environments

A specialized development environment simplifies web services migration by automating many of the tasks. For development of WebSphere-based applications, including web services, you can use the assembly tools provided with WebSphere Application Server.

The following sample code in this topic was tested in the following runtime environment:

- IBM WebSphere Application Server V6.1, which includes support for the JAX-RPC specification.

- IBM WebSphere Application Server V6.1 Feature Pack for Web Services, which includes support for the JAX-WS and JAXB specifications.

Examples

The following examples show some of the code changes that you might need to migrate your JAX-RPC Web services to JAX-WS and JAXB web services. In particular, there is an emphasis on the changes in the implementation code that you must write, from both the client and the server side. If no new function is to be introduced, the changes required to move from JAX-RPC to JAX-WS and JAXB might be few.

Sample web service

The first example is a sample JAX-RPC-based web service that was created from a WSDL file (top-down development); the same WSDL file is used to generate the JAX-WS and JAXB-based service. The WSDL file describes the web service, SampleService, with these operations described by the following image:

As shown in the image, the five operations offered are:

- xsd:int calcShippingCost(xsd:int, xsd:int)
- xsd:string getAccountNumber(xsd:string)
- xsd:dateTime calculateShippingDate(xsd:dateTime)
- xsd:string ckAvailability(xsd:int) creates invalidDateFault
- Person findSalesRep(xsd:string)

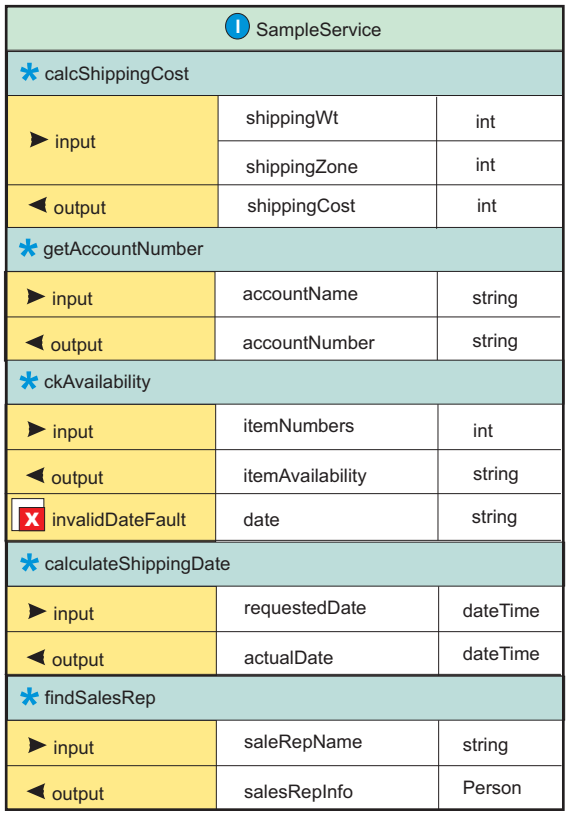
Also assume that Person is defined by the following schema fragment in the WSDL file:

```
<complexType name="Person">
 <sequence>
  <element name="name" type="xsd:string"/>
  <element name="age" type="xsd:int"/>
  <element name="location" type="impl:Address/>
 </sequence>
</complexType>
```

Address is defined by:

```
<complexType name="Address">
 <sequence>
  <element name="street" type="xsd:string"/>
  <element name="city" type="xsd:string"/>
  <element name ="state" type="xsd:string"/>
  <element name="zip" type="xsd:int"/>
 </sequence>
</complexType>
```

The following image also shows that the operation, ckAvailability(xsd:int), which produces an invalidDateFault exception.



Service operations

Review the service code created by the tooling. The following information includes examining what is created for a JAX-RPC runtime and also for a JAX-WS and JAXB runtime.

JAX-RPC

For JAX-RPC, the tooling accepts the WSDL file as input, and, amongst other files, generates the SampleService.java and SampleServiceImpl.java interfaces. The SampleService.java interface defines an interface and the generated code can be review in the following code block. The SampleServiceSoapBindingImpl.java interface provides the skeleton of an implementation, and you typically modify to add your own logic.

JAX-RPC version of SampleService.java:

```
/**
* SampleService.java
*
* This file was auto-generated from WSDL
* by the IBM web services WSDL2Java emitter.
* cf20633.22 v82906122346
*/
package simple;
public interface SampleService extends java.rmi.Remote {
 public java.lang.Integer calcShippingCost(java.lang.Integer shippingWt,
  java.lang.Integer shippingZone) throws java.rmi.RemoteException;
 public java.lang.String getAccountNumber(java.lang.String accountName)
  throws java.rmi.RemoteException;
 public java.lang.String[] ckAvailability(int[] itemNumbers)
  throws java.rmi.RemoteException, simple.InvalidDateFault;
 public java.util.Calendar calculateShippingDate(
  java.util.Calendar requestedDate)
  throws java.rmi.RemoteException;
 public simple.Person findSalesRep(java.lang.String saleRepName)
  throws java.rmi.RemoteException; }
```

JAX-WS and JAXB

For JAX-WS and JAXB, the tooling accepts the WSDL file as input, and as in the case of JAX-RPC, generates SampleService.java and SampleServiceImpl.java interfaces. As with JAX-RPC, the SampleService.java interface also defines an interface as shown in the following code block. The SampleServiceImpl.java interface provides the skeleton of an implementation, and you typically modify to add your own logic.

Attention: The code has been annotated by the tooling.
JAX-WS and JAXB version of the SampleService.java interface:

```
package simple;
 import java.util.List;
 import javax.jws.WebMethod;
 import javax.jws.WebParam;
 import javax.jws.WebResult;
 import javax.jws.WebService;
 import javax.xml.datatype.XMLGregorianCalendar;
 import javax.xml.ws.RequestWrapper;
 import javax.xml.ws.ResponseWrapper;

/**
* This class was generated by the JAX-WS SI.
* JAX-WS RI IBM 2.0_03-06/12/2007 07:44 PM(Raja)-fcs
* Generated source version: 2.0
*
*/
@WebService(name = "SampleService", targetNamespace = "http://simple") public interface SampleService {

 /**
 *
 * @param shippingWt
 * @param shippingZone
 * @return
 * returns java.lang.Integer
 */
 @WebMethod
 @WebResult(name = "shippingCost", targetNamespace = "")
 @RequestWrapper(localName = "calcShippingCost", targetNamespace = "http://simple",
className = "simple.CalcShippingCost")
 @ResponseWrapper(localName = "calcShippingCostResponse", targetNamespace = "http://simple",
className = "simple.CalcShippingCostResponse")
 public Integer calcShippingCost(
 @WebParam(name = "shippingWt", targetNamespace = "")
 Integer shippingWt,
 @WebParam(name = "shippingZone", targetNamespace = "")
 Integer shippingZone);

 /**
 *
 *
 @param accountName
 * @return
 *     returns java.lang.String
 */
 @WebMethod
 @WebResult(name = "accountNumber", targetNamespace = "")
 @RequestWrapper(localName = "getAccountNumber", targetNamespace = "http://simple",
className = "simple.GetAccountNumber")
 @ResponseWrapper(localName = "getAccountNumberResponse", targetNamespace = "http://simple",
className = "simple.GetAccountNumberResponse")
 public String getAccountNumber(
  @WebParam(name = "accountName", targetNamespace = "")
  String accountName);
 /**
 *
 * @param requestedDate
 * @return
 *     returns javax.xml.datatype.XMLGregorianCalendar
 */
 @WebMethod
 @WebResult(name = "actualDate", targetNamespace = "")
 @RequestWrapper(localName = "calculateShippingDate", targetNamespace = "http://simple",
className = "simple.CalculateShippingDate")
 @ResponseWrapper(localName = "calculateShippingDateResponse", targetNamespace = "http://simple",
```

```
className = "simple.CalculateShippingDateResponse")
 public XMLGregorianCalendar calculateShippingDate(
  @WebParam(name = "requestedDate", targetNamespace = "")
  XMLGregorianCalendar requestedDate);
 /**
 *
 * @param itemNumbers
 * @return
 *     returns java.util.List<java.lang.String>
 * @throws InvalidDateFault_Exception
 */
 @WebMethod
 @WebResult(name = "itemAvailability", targetNamespace = "")
 @RequestWrapper(localName = "ckAvailability", targetNamespace = "http://simple", className = "simple.CkAvailability")
 @ResponseWrapper(localName = "ckAvailabilityResponse", targetNamespace = "http://simple",
className = "simple.CkAvailabilityResponse")
 public List<String> ckAvailability(
  @WebParam(name = "itemNumbers", targetNamespace = "")
  List<Integer> itemNumbers)
  throws InvalidDateFault_Exception
 ;
 /**
 *
 * @param saleRepName
 * @return
 *     returns simple.Person
 */
 @WebMethod
 @WebResult(name = "salesRepInfo", targetNamespace = "")
 @RequestWrapper(localName = "findSalesRep", targetNamespace = "http://simple", className = "simple.FindSalesRep")
 @ResponseWrapper(localName = "findSalesRepResponse", targetNamespace = "http://simple",
className = "simple.FindSalesRepResponse")
 public Person findSalesRep(
 @WebParam(name = "saleRepName", targetNamespace = "")
 String saleRepName);

}
```

Comparing the code examples

At first glance, it might seem that there is little similarity between the interfaces. If you disregard the additional information added by the annotations for JAX-WS and JAXB, the code samples are similar. For example, the calcShippingCost method in the JAX-WS and JAXB version the following lines of code exist:

```
@WebMethod
 @WebResult(name = "shippingCost", targetNamespace = "")
 @RequestWrapper(localName = "calcShippingCost", targetNamespace = "http://simple",
className = "simple.CalcShippingCost")
 @ResponseWrapper(localName = "calcShippingCostResponse", targetNamespace = "http://simple",
className = "simple.CalcShippingCostResponse")
 public Integer calcShippingCost(
  @WebParam(name = "shippingWt", targetNamespace = "")
  Integer shippingWt,
  @WebParam(name = "shippingZone", targetNamespace = "")
  Integer shippingZone);
```

But, if you discard the annotations, the following lines of code are:

```
public Integer calcShippingCost(
        Integer shippingWt,
        Integer shippingZone);
```

These lines are almost identical to what was generated for JAX-RPC. The only difference is that the JAX-RPC code can produce the java.rmi.RemoteException error as follows:

```
public java.lang.Integer calcShippingCost(java.lang.Integer shippingWt,
   java.lang.Integer shippingZone) throws java.rmi.RemoteException;
```

Following this logic, three of the methods have essentially the same signatures:

```
public Integer calcShippingCost(Integer shippingWt, Integer shippingZone)
 public String getAccountNumber(String accountName)
 public Person findSalesRep(String saleRepName)
```

This means that migrating from JAX-RPC to JAX-WS does not directly affect these methods and the original implementation code that is successfully running in the JAX-RPC based environment can probably be used without modification for these methods.

However two of the methods do have different signatures:

For JAX-RPC:

```
public java.util.Calendar calculateShippingDate(
 java.util.Calendar requestedDate)
public java.lang.String[] ckAvailability(int[] itemNumbers)
 throws java.rmi.RemoteException,
 simple.InvalidDateFault
```

JAX-WS and JAXB:

```
public XMLGregorianCalendar calculateShippingDate(
 XMLGregorianCalendar requestedDate)
public List<String> ckAvailability(List<Integer> itemNumbers)
 throws InvalidDateFault_Exception
```

Attention:

You might find it easier to compare the skeleton implementation files since the annotations are not present in SampleServiceImpl.java.:

- SampleServiceSoapBindingImpl.java
- SampleServiceImpl.java

The differences in signatures are due to the following reasons:

- Differences mappings from XML names to Java names

  For the calculateShippingDate method, both the input parameter and the return parameter have changed from type java.util.Calendar to type XMLGregorianCalendar. This is because the WSDL specified these parameters to be of type, xsd:dateTime, JAX-RPC maps this data type to java.util.Calendar, while JAX-WS and JAXB maps it to XMLGregorianCalendar.
- Different mappings of arrays from XML to Java

  For the ckAvailability method, the change is due to the data mappings for XML arrays.

  JAX-RPC maps the following WSDL elements:

```
<element maxOccurs="unbounded" name="itemNumbers" type="xsd:int"/>
<element maxOccurs="unbounded" name="itemAvailability" type="xsd:string"/>
```

  The previous elements are mapped to the following Java source:

```
int[] itemNumbers
java.lang.String[] itemAvailability
```

  JAX-WS and JAXB map the following WSDL elements:

```
List<Integer> itemNumbers
List<String> ckAvailability
```

- Different mappings of exceptions

  For the ckAvailability method, the JAX-RPC code generated the following error:

```
simple.InvalidDateFault
```

  The JAX-WS code generates the following error:

```
InvalidDateFault_Exception
```

  Except for the names, the constructors for these exceptions are different. Therefore, the actual JAX-RPC code that produces the error might be displayed as:

```
throw new InvalidDateFault("this is an InvalidDateFault");
```

For JAX-WS, the code uses a command similar to the following example::

```
throw new InvalidDateFault_Exception( "this is an InvalidDateFault_Exception", new InvalidDateFault());
```

In cases that use exceptions, the code that uses it needs to change.

Migrating the code

Now that the differences between code have been explained, review the original code for the methods that need changing and how to change this code so that it works in a JAX-WS and JAXB environment.

Assume that the original (JAX-RPC) implementation code is similar to following:

```
public java.util.Calendar calculateShippingDate(
     java.util.Calendar requestedDate) throws java.rmi.RemoteException {
// Set the date to the date that was sent to us and add 7 days.
requestedDate.add(java.util.Calendar.DAY_OF_MONTH, 7);

// . . .

return requestedDate;
}
```

You can write the new code to directly use the new types as in the following examples:

```
public XMLGregorianCalendar calculateShippingDate(
XMLGregorianCalendar requestedDate) {
try {
// Create a data type factory.
DatatypeFactory df = DatatypeFactory.newInstance();
// Set the date to the date that was sent to us and add 7 days.
Duration duration = df.newDuration("P7D");
requestedDate.add(duration);

} catch (DatatypeConfigurationException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

// . . .

return requestedDate;
}
```

Migrating the code for the ckAvailability method

When migrating the code for the ckAvailability method, changes were required because of the way that arrays and exceptions are mapped from XML is different between JAX-RPC and JAX-WS. Assume that the original JAX-RPC web service code is similar to the following example:

```
public java.lang.String[] ckAvailability(int[] itemNumbers)
{

    String[] myString = new String[itemNumbers.length];
    for (int j = 0; j < myString.length; j++) {

            . . .
             if ( ... )

    throw new simple.InvalidDateFault("InvalidDateFault");

            . . .
            if ( . . .)
               myString[j] = "available:  " + jitemNumbers[j] ;
            else
               myString[j] = "not available:  " + jitemNumbers[j];

    }
    return myString;
}
```

The previous code accepts an int[] as input and returns a String[]. For the JAX-WS and JAXB version, these are List<Integer> and List<String> elements respectively. Processing for these arrays, and discarding the Exception code, the JAX-RPC code is similar to the following:

```
public java.lang.String[] ckAvailability(int[] itemNumbers)
{

    String[] myString = new String[itemNumbers.length];
    for (int j = 0; j < jitemNumbers.length; j++) {


            . . .
            if ( . . .)
               myString[j] = "available:  " + itemNumbers.get(j);
            else
               myString[j] = "not available:  " + itemNumbers.get(j);
    }
    return myString;
}
```

The following JAX-WS and JAXB equivalent code exists using Lists instead of arrays:

```
List <String> ckAvailability(List <Integer> itemNumbers)
{

    ArrayList<String> retList = new ArrayList<String>();
    for (int count = 0; count < itemNumbers.size(); count++) {


            . . .
            if ( . . .)
               retList.add("available:  " + itemNumbers.get(j));
            else
               retList.add("not available:  " + itemNumbers.get(j));
    }
    return retList;
}
```

The differences in the mappings of exceptions from XML to Java forces the JAX-WS code to use. InvalidDateFault_Exception instead of InvalidDateFault.

This means that you must replace `throw new simple.InvalidDateFault("InvalidDateFault");` with some other code. Therefore, the following line is used for the new code:

```
throw new InvalidDateFault_Exception( "test InvalidDateFault_Exception", new InvalidDateFault());
```

The final JAX-WS and JAXB implementation of the method might be similar to the following code:

```
List <String> ckAvailability(List <Integer> itemNumbers)
{

    ArrayList<String> retList = new ArrayList<String>();
    for (int count = 0; count < itemNumbers.size(); count++) {

if ( . . . ) {
   throw new InvalidDateFault_Exception(
   "test InvalidDateFault_Exception",
   new InvalidDateFault());
}

. . .
if ( . . .)
   retList.add("available:  " + itemNumbers.get(count));
else
   retList.add("not available:  " + itemNumbers.get(count));
   }
   return retList;
}
```

There are multiple ways that you might choose to migrate the code. With practice and an effective development environment, migrating from JAX-RPC to JAX-WS can be straightforward.

Web services migration best practices

Use these web services migration best practices when migrating web services applications.

If you have used the Apache SOAP support to develop web services client applications in WebSphere Application Server Versions 4, 5, or 5.1, you might need to migrate your applications or the security files for your applications. The following table summarizes the web services specifications supported by the WebSphere products.

Table 6. Summary of web services specifications supported by WebSphere Application Server. The product supports the web services specifications in this table.

WebSphere Application Server Version	Web services specifications supported
4.0	Apache SOAP 2.2
5.0 and 5.0.1	Apache SOAP 2.3
5.0.2 or later	Java 2 Platform, Enterprise Edition (J2EE), also known as (JSR 109)
6.0.x and 6.1	J2EE (JSR 109)
7.0 or later	Web Services for Java Platform, Enterprise Edition (Java EE) 5 also known as JSR 109

Note: The Apache SOAP 2.2 and Apache SOAP 2.3-based implementations that were available in WebSphere Application Server Version 4.0.*x*, 5.0 and 5.0.1 are no longer supported. It is recommended that applications that are using these SOAP implementations migrate to Web Services for Java EE (JSR 109) support that is provided in current WebSphere Application Server versions.

For more information on migrating your web services, read about migrating Apache SOAP web services to web services to J2EE standards.

It is recommended that new web services be developed using the web services for Java EE specification. To learn more, read about implementing web services applications.

Security cannot be directly migrated from SOAP 2.3 to the Java EE standards. After you have migrated your web services to the Java EE standards, consider securing your web services applications. To learn more, read about securing web services applications using message level security.

Follow these best practices for the most optimal migration experience:

The 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.

Note: Existing JAX-RPC applications wanting to use JAX-WS features must be rewritten using the JAX-WS programming model.

Redeploy existing JAX-RPC web services after migrating to a new release of the application server

When migrating to a new release of the application server, it is recommended that you redeploy your web services applications. You should redeploy your web services application in the new application server environment because of possible changes to the supported levels of web services specifications and web services deployment descriptors in each release. To redeploy your web service, select **Deploy Web Services** in the Install New Application wizard or use the wsdeploy command. To learn more about this process, see the deploying web services applications onto application servers documentation.

Migrating a Version 5 Java API for XML-based remote procedure call (JAX-RPC) client that uses SOAP over Java Message Service (JMS) to invoke a web service

A JAX-RPC client that is run on WebSphere Application Server Version 5, can use SOAP over JMS to invoke a web service that is run on a Version 5 Application Server.

A user ID and password are not required on the target WebSphere MQ queue. After the application server is migrated to Version 6.*x*, and uses the Version 6.*x* default messaging feature, client requests can fail because basic authentication is enabled. The following error message displays when this migration problem occurs:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server with
endpoint <endpoint name> in bus <bus_name> with reason: CWSIT0016E: The user
ID null failed authentication in bus <bus_name>.
```

When the application server is migrated to Version 6.*x*, and the default messaging provider (service integration technologies) is used, and administrative and application security is enabled for the server or the cell, the service integration bus queue destination inherits the security characteristics of the server or the cell by default. If the server or the cell has basic authentication enabled, the client request fails.

The following options are available to solve this problem. The solutions are listed by the level of security that they impose:

- Disable administrative and application security on the main security panel within the administrative console. To disable administrative and application security, click **Security** > **Global security**. Deselect the **Enable administrative security** and **Enable application security** options.
- Modify the settings for the service integration bus that hosts the queue destination so that the bus security is disabled and the bus does not inherit security characteristics from the server or the cell. This option is equivalent to the level of security that you can configure in Version 5.
- Configure the basic authentication on each client that uses the service. To learn more, see the information on configuring HTTP basic authentication for JAX-RPC web services with the administrative console.

Migrating Apache SOAP web services

You can migrate web services that were developed using Apache SOAP to web services that are developed based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification. See the information on migrating Apache SOAP web services to JAX-RPC web services based on Java EE standards.

Migrating web services assembled with early versions of the Application Server Toolkit or Assembly Toolkit

If you are migrating your web service or web service components from earlier versions of the Application Server Toolkit or Assembly Toolkit, refer to the following hints and tips to improve your success:

- Secure web services are not migrated by the J2EE Migration Wizard when web services are migrated from J2EE 1.3 to J2EE 1.4.
- The migration of secure web services requires manual steps.
- After the J2EE migration, the secure binding and extension files must be migrated manually to J2EE 1.4 as follows:
  1. Double click on the `webservices.xml` file to open the web services editor.
  2. Select the **Binding Configurations** tab to edit the binding file.
  3. Add all the necessary binding configurations under the new sections **Request Consumer Binding Configuration Details** and **Response Generator Binding Configuration Details**.
  4. Select the **Extension** tab to edit the extension file.

5. Add all the necessary extension configurations under the new sections **Request Consumer Service Configuration Details** and **Response Generator Service Configuration Details**.
6. Save and exit the editor.

.

Migrating Apache SOAP web services to JAX-RPC web services based on Java EE standards

You can 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.

Before you begin

If you have used web services based on Apache SOAP and now want to develop and implement web services that are Java-based, you need to migrate client applications developed with all versions of 4.0, and versions of 5.0 prior to 5.0.2.

About this task

Note: For this version of the application server, SOAP-Security (XML digital signature) based on Apache SOAP implementation has been removed. Instead of using SOAP-Security, migrate and re-configure your application to the JSR-109 implementation of web services.

To migrate these client applications according to Java standards:

Procedure

1. Plan your migration strategy. You can port an Apache SOAP client to a JAX-RPC web services client in one of two ways:
   - If you have, or can create, a Web Services Description Language (WSDL) document for the service, consider using the WSDL2Java command tool to generate bindings for the web service. It is more work to adapt an Apache SOAP client to use the generated JAX-RPC bindings, but the resulting client code is more robust and easier to maintain.

     To follow this path, see the article on developing a web services client in the information center.
   - If you do not have a WSDL document for the service, do not expect the service to change, and you want to port the Apache SOAP client with minimal work, you can convert the code to use the JAX-RPC dynamic invocation interface (DII), which is similar to the Apache SOAP APIs. The DII APIs do not use WSDL or generated bindings.

   Because JAX-RPC does not specify a framework for user-written serializers, the JAX-RPC does not support the use of custom serializers. If your application cannot conform to the default mapping between Java, WSDL, and XML technology supported by WebSphere Application Server, do not attempt to migrate the application. The remainder of this topic assumes that you decided to use the JAX-RPC dynamic invocation interface (DII) APIs.
2. Review the GetQuote Sample. A web services migration Sample is available in the Samples section of the Information Center. This sample is located in the `GetQuote.java` file, originally written for Apache SOAP users, and includes an explanation about the changes needed to migrate to the JAX-RPC DII interfaces. To learn more, see the Samples section of the Information Center.
3. Convert the client application from Apache SOAP to JAX-RPC DII The Apache SOAP API and JAX-RPC DII API structures are similar. You can instantiate and configure a call object, set up the parameters, invoke the operation, and process the result in both. You can create a generic instance of a Service object with the following command:

```
javax.xml.rpc.Service service = ServiceFactory.newInstance().createService(new QName(""));
```

in JAX-RPC.

a. Create the Call object. An instance of the Call object is created with the following code:

```
org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call ()
```

in Apache SOAP.

An instance of the Call object is created by

```
java.xml.rpc.Call call = service.createCall();
```

in JAX-RPC.

b. Set the endpoint Uniform Resource Identifiers (URI). The target URI for the operation is passed as a parameter to

```
call.invoke:  call.invoke("http://...", "");
```

in Apache SOAP.

The setTargetEndpointAddress method is used as a parameter to

```
call.setTargetEndpointAddress("http://...");
```

in JAX-RPC.

Apache SOAP has a setTargetObjectURI method on the Call object that contains routing information for the request. JAX-RPC has no equivalent method. The information in the targetObjectURI is included in the targetEndpoint URI for JAX-RPC.

c. Set the operation name. The operation name is configured on the Call object by

```
call.setMethodName("opName");
```

in Apache SOAP.

The setOperationName method, which accepts a `QName` instead of a `String` parameter, is used in JAX-RPC as illustrated in the following example:

```
call.setOperationName(new javax.xml.namespace.Qname("namespace", "opName"));
```

d. Set the encoding style. The encoding style is configured on the Call object by

```
call.setEncodingStyleURI(org.apache.soap.Constants.NS_URI_SOAP_ENC);
```

in Apache SOAP.

The encoding style is set by a property of the Call object

```
call.setProperty(javax.xml.rpc.Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.
xmlsoap.org/soap/encoding/");
```

in JAX-RPC.

e. Declare the parameters and set the parameter values. Apache SOAP parameter types and values are described by parameter instances, which are collected into a vector and set on the Call object before the call, for example:

```
Vector params = new Vector ();
params.addElement (new org.apache.soap.rpc.Parameter(name, type, value, encodingURI));
// repeat for additional parameters...
call.setParams (params);
```

For JAX-RPC, the Call object is configured with parameter names and types without providing their values, for example:

```
call.addParameter(name, xmlType, mode);
// repeat for additional parameters
call.setReturnType(type);
```

Where

- *name* (type `java.lang.String`) is the name of the parameter
- *xmlType* (type `javax.xml.namespace.QName`) is the XML type of the parameter

- *mode* (type `javax.xml.rpc.ParameterMode`) the mode of the parameter, for example, IN, OUT, or INOUT

f. Make the call. The operation is invoked on the Call object by

```
org.apache.soap.Response resp = call.invoke(endpointURI, "");
```

in Apache SOAP.

The parameter values are collected into an array and passed to `call.invoke` as illustrated in the following example:

```
Object resp = call.invoke(new Object[] {parm1, parm2,...});
```

in JAX-RPC.

g. Check for faults. You can check for a SOAP fault on the invocation by checking the response:

```
if resp.generatedFault then {
org.apache.soap.Fault f = resp.getFault;
f.getFaultCode();
f.getFaultString();
}
```

in Apache SOAP.

A `java.rmi.RemoteException` error is displayed in JAX-RPC if a SOAP fault occurs on the invocation.

```
try {
... call.invoke(...)
} catch (java.rmi.RemoteException) ...
```

h. Retrieve the result. In Apache SOAP, if the invocation is successful and returns a result, it can be retrieved from the Response object:

```
Parameter result = resp.getReturnValue(); return result.getValue();
```

In JAX-RPC, the result of invoke is the returned object when no exception is displayed:

```
Object result = call.invoke(...);
 ...
return result;
```

Results

You have migrated Apache SOAP web services to a JAX-RPC Web services based on the Java EE specification.

What to do next

Develop a web services client. See the article in the information center on how to develop a Web services client based on the Web Services for Java EE specification.

Test the web services-enabled clients to make sure that the migration process is successful and you can implement the web services in a Java EE environment.

Migrating Web Services Security

You can migrate Web Services Security bindings from an older version to the latest version of WebSphere® Application Server. The product migration function handles most of the migration process, but your input and action is required for specific configurations in order to complete the migration.

Migration of JAX-WS Web Services Security bindings from Version 6.1

You can migrate Web Services Security bindings from an older version to Version 7.0 and later of WebSphere Application Server. Migration of the JAX-WS bindings in Version 6.1 Feature Pack for Web Services takes place during the product migration to Version 7.0 and later.

The product migration handles most of the WS-Security migration process, but your input and action is required for specific configurations in order to complete the migration. Following are some examples of configurations that require manual migration steps:

- Using callers on Version 6.1 Feature Pack for Web Services.

  WebSphere Application Server Version 7.0 and later supports the capability to specify a preference order for callers. In the situation where only a single caller is present in the bindings, product migration automatically assigns an order of **1** to the single caller that is present. However, if there are multiple callers, warnings are logged during migration, and you must manually assign the caller preference order. Set the order attribute for each caller using the administrative console, or the administration commands, after product migration is completed. Read about call collection and configuring the callers for general and default bindings for instructions on setting the caller order using the administrative console. See the topic "WS-Security policy and binding properties" for information on using the administration commands.

- Using multiple username tokens in the default bindings.

  During product migration for Version 6.1 default bindings, all UsernameToken generators and consumers in the bindings will be migrated. However, if multiple username token generators, or consumers, are used, a warning is logged during migration. The warning states that a maximum of two username token generators and two username token consumers are allowed, and that one of each pair of token generators or consumers must have the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property set to true. You must manually select which username token consumer or generator to keep, and which token has the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property, using the administrative console, or administration commands.

  – To use the administrative console to set the property, access the WS-Security 6.1 default bindings as instructed in the "Policy set bindings settings for WS-Security" topic, and click the **Authentication and Protection** link. Add, remove or edit the username token consumers and generators as needed, following the instructions in the "WS-Security authentication and protection for general bindings" topic.

  – To use the administration commands to set the property, and to add, delete and edit the username token generators or consumers as needed, refer to the "WS-Security policy and bindings properties" topic for instructions.

- Using multiple token generators and token consumers of the same type in Version 6.1 default bindings.

  During product migration for Version 6.1 default bindings, all token generators and token consumers for supporting tokens are migrated. In the situation where multiple token generators and token consumers of the same supporting token type are found a warning is logged during migration. This warning states that only a single token consumer and token generator for each supporting token type must be present. You must manually select which token consumer or generator to use for the repeating supporting token type, using the administrative console or administration commands. To use the administrative console to select the token, access the WS-Security 6.1 default bindings as instructed in the "Policy set bindings settings for WS-Security" topic, and select the **Authentication and Protection** link. Add, edit or remove token consumers and generators as needed, following the instructions in the "WS-Security authentication and protection for general bindings" topic.

Migrating JAX-RPC Web Services Security applications to Version 8.0 applications

Migration of a Java Platform, Enterprise Edition (Java EE) Version 1.3 application that uses Web Services Security to a Java EE Version 1.4 application is possible.

Before you begin

You can install Java Platform, Enterprise Edition (Java EE) Version 1.3 applications that use Web Services Security on a WebSphere Application Server Version 8.0 server. However, if you want Java EE Version 1.3 applications to use the Web Services Security (WSS) Version 1.0 or 1.1 specifications and the other new

features added in Version 8.0, you must migrate the Java EE Version 1.3 applications to Java EE Version 1.4.

About this task

Complete the following steps to migrate a Version 1.3 application, along with the Web Services Security configuration information, to a Version 8.0 application:

Procedure

1. Save the original Java EE Version 1.3 application. You need the Web Services Security configuration files of the Java EE Version 1.3 application to recreate the configuration in the new format for the Java EE Version 1.4 application.
2. Use the Java Platform, Enterprise Edition (Java EE) Migration Wizard in an assembly tool to migrate the Java EE Version 1.3 application to Java EE Version 1.4.

   Important: After you migrate to Java EE Version 1.4 using the Java EE Migration Wizard, you cannot view the Java EE Version 1.3 extension and binding information within an assembly tool. You can view the Java EE Version 1.3 Web Services Security extension and binding information using a text editor. However, do not edit the extension and binding information using a text editor. The Java EE Migration Wizard does not migrate the Web Services Security configuration files to the new format in the Java EE Version 1.4 application. Rather the wizard is used to migrate your files from Java EE Version 1.3 to Version 1.4.

   To access the Java EE Migration Wizard, complete the following steps:

   a. Right-click the name of your application.
   b. Click **Migrate** > **Java EE Migration Wizard**.
3. Manually delete all of the Web Services Security configuration information from the binding and extension files of the application that is migrated to Java EE Version 1.4.
   a. Delete the `<securityRequestReceiverServiceConfig>` and `<securityResponseSenderServiceConfig>` sections from the server-side `ibm-webservices-ext.xmi` extension file.
   b. Delete the `<securityRequestReceiverBindingConfig>` and `<securityResponseSenderBindingConfig>` sections from the server-side `ibm-webservices-bnd.xmi` binding file.
   c. Delete the `<securityRequestSenderServiceConfig>` and `<securityResponseReceiverServiceConfig>` sections from the client-side `ibm-webservicesclient-ext.xmi` extension file.
   d. Delete the `<securityRequestSenderBindingConfig>` and `<securityResponseReceiverBindingConfig>` sections from client-side `ibm-webservicesclient-bnd.xmi` binding file.
4. Recreate the Web Services Security configuration information in the new Java EE Version 1.4 format. At this stage, because the application is already migrated to the Java EE Version 1.4, use an assembly tool to configure the original Web Services Security information in the new Version 8.0 format. For more information on assembly tools, see the related information.

Results

This task provides general information about how to migrate Java EE Version 1.3 applications to Java EE Version 1.4.

What to do next

The following articles contain some general scenarios that map some of the basic Web Services Security information specified in a Java EE Version 1.3 application to a Java EE Version 1.4 application and specify

this information using an assembly tool. The Web Services Security configuration information is contained in four configuration files: two server-side configuration files and two client-side configuration files. The migration of all of the configuration information is divided into four sections; one for each configuration file. When you recreate the Web Services Security information in the new Java EE Version 1.4 format, it is recommended that you configure the extensions and binding files in the following order:

1. Configure the `ibm-webservices-ext.xmi` server-side extensions file. For more information, see "Migrating the JAX-RPC server-side extensions configuration."
2. Configure the `ibm-webservicesclient-ext.xmi` client-side extensions file. For more information, see "Migrating the client-side extensions configuration" on page 95.
3. Configure the `ibm-webservices-bnd.xmi` server-side bindings file. For more information, see "Migrating the server-side bindings file" on page 96.
4. Configure the `ibm-webservicesclient-bnd.xmi` client-side bindings file. For more information, see "Migrating the client-side bindings file" on page 98.

Migrating the JAX-RPC server-side extensions configuration

You can migrate the Web Services Security server-side extensions configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application for the JAX-RPC programming model.

About this task

The following table lists the mappings for the top-level sections under the server-side **Security Extensions** tab within an assembly tool from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 7. The mapping of the configuration sections. Use the extensions configuration information for migration.

Java EE Version 1.3 extensions configuration	Java EE Version 1.4 extensions configuration
Request Receiver Service Configuration Details	Request Consumer Service Configuration Details
Response Sender Service Configuration Details	Response Generator Service Configuration Details

For information about the assembly tools that are available for WebSphere Application Server Version 6.0.*x*, see the assembly tools information.

Consider the following steps to migrate the server-side extensions from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration.

Procedure

- Import the Java EE Version 1.3 application into an assembly tool and identify all the message parts that are required to be signed and encrypted. The message parts are listed in the Required Integrity and Required Confidentiality sections under the Request Receiver Service Configuration Details section. In a Java EE Version 1.4 application, these message parts map to the Message parts field of the **Required integrity** and **Required confidentiality** dialogs windows within the assembly tool.

  To specify these message parts within an assembly tool, complete the following steps in the Web Services Editor. The steps are based on typical scenarios, but the steps are not all-inclusive.

  1. Click the **Extensions** tab.
  2. Navigate to the Required integrity subsection within the Request Consumer Service Configuration Details section.
  3. Specify each message part to be signed in the Message Parts field.

  For example, if the message part in the Java EE Version 1.3 application is `body`, you need to specify `body` in the Message parts keyword field. Similarly, on the **Extensions** tab, configure the message parts to be encrypted using the Required Confidentiality dialog. Also, for all the message parts that are

migrated from a Java EE Version 1.3 application, you must select **http://www.ibm.com/websphere/webservices/wssecurity/dialect-was** in the Message parts dialect field and **Required** in the Usage type field.

- Optional: Configure the Required Security Token and Caller Part sections on the **Extensions** tab if the authentication method of BasicAuth is configured under the Login Config section of the Java EE Version 1.3 application. When you configure the Required Security Token section, select **Username** in the name field and **Required** in the Usage type field within the Required Security Token Dialog window. The following table shows how the authentication method values for a Java EE Version 1.3 application map to the token type values within the Java EE Version 1.4 application.

Table 8. Authentication method to token type mappings. Use the extensions configuration information for migration.

Login Config Authentication method values in the Java EE Version 1.3 extensions configuration	Token type values in the Java EE Version 1.4 extensions configuration
`BasicAuth`	`UsernameToken`
`Signature`	`X509 certificate`
`LTPA`	`LTPAToken`

If the authentication method value is IDAssertion within the Login Config section, the token type that you must specify in the Java EE Version 1.4 application depends upon the IDType value within the IDAssertion section. The following table shows how the IDType values for Java EE Version 1.3 application map to the token type values in the Java EE Version 1.4 application.

Table 9. IDType values to token type mappings. Use the extensions configuration information for migration.

IDType values in the Java EE Version 1.3 application extensions configuration	Token type values in the Java EE Version 1.4 application extensions configuration
`X509Certificate`	`X509 certificate`
`Username`	`Username`

- Select the appropriate token type in the Name field of the Call Part Dialog window based on the previous two tables. Select the `Username` token type when you are configuring the caller part for the basic authentication method. Configuring the other token types in the Caller part dialog is similar to configuring token types in the Required Security Token dialog. If you need to map the IDAssertion authentication method from a Java EE Version 1.3 application to a Java EE Version 1.4 application, select the **Use IDAssertion** option and configure the ID assertion section of the Caller Part Dialog window. The Trust Mode field under the IDAssertion section maps to the Trust method name field of the Trust method property section in the Caller Part Dialog window. If Signature is selected for the Trust method, specify the Required Integrity part that specifies the signature of the trusted intermediary certificate.
- Configure a nonce in the Version 8.0 Binding Configurations section if nonce is specified in the Add Authentication Method dialog under Login Config within the Java EE Version 1.3 application extensions configuration.

  Important: Nonce is configured in the bindings for a Java EE Version 1.4 application and not in the extensions.

  To configure a nonce on the Binding Configurations tab, set the `com.ibm.wsspi.wssecurity.token.username.verifyNonce` property in the Token Consumer configuration for the Username token.
- Configure the Add Timestamp section to migrate the time stamp information if the <addReceivedTimestamp> element is configured in the Java EE Version 1.3 extensions. To migrate the Response Sender Service Configuration Details section in the Java EE Version 1.3 extensions, identify all of the message parts listed within the Integrity and Confidentiality sections. Configure these message parts using the Integrity and Confidentiality dialogs under the Response Generator Service Configuration details section. This configuration is similar to the configuration for Required Integrity and Required Confidentiality, with the exception of the Order field in the Integrity Dialog. The value of this Order field specifies the order in which the message parts specified in the Message Parts field are digitally signed or encrypted in the SOAP message. For example, the extensions contain the following information:

– One integrity entry called `int_part1` with a value of `1` in the Order field
– One confidentiality entry called `conf_part1` with a value of `2` in the Order field

In this example, the message parts that are specified by the `int_part1` integrity entry are signed before the message parts specified by the `conf_part1` confidentiality entry are encrypted. The same rule for the order attribute applies for multiple integrity or confidentiality elements.

Results

These steps describe the types of information that you need to migrate the Web Services Security server-side extensions for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the client-side extensions for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the client-side extensions configuration.”

Migrating the client-side extensions configuration

You can migrate the Web Services Security client-side extensions configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mappings of the top-level sections under the client-side **Security Extensions** tab for Web Services Security from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 10. The mapping of the configuration sections. Use the extensions configuration information for migration.

Java EE Version 1.3 security extensions for Web Services Security	Java EE Version 1.4 extensions for Web Services Security
Request Sender Configuration	Request Generator Configuration
Response Receiver Configuration	Response Consumer Configuration

Consider the following steps to migrate the client-side extensions configuration from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

Procedure

- Migrate the message parts that you need to sign or encrypt from the Integrity and Confidentiality sections in the Java EE Version 1.3 application to the Integrity and Confidentiality sections on the **WS Extensions** tab in an assembly tool for a Java EE Version 1.4 application.
- Configure the Security Token section under the Request Generator Configuration on the **WS Extensions** tab if Login Config section is configured in the Java EE Version 1.3 extensions configuration. When you configure the security token, select the token type in the Token type field that matches the authentication method value of the Login Config in the Java EE Version 1.3 application. For example, if the authentication method in the Java EE Version 1.3 extensions configuration is BasicAuth, then select **Username** in the Token type field within the assembly tool. For more information on how the authentication methods for Web Services Security map from a Java EE Version 1.3 application to a Java EE Version 1.4 application, see Table 8 on page 94. If the authentication method is IDAssertion, there is no action required because in a Java EE Version 1.4 application the identity assertion configuration is not required in the client-side extensions configuration. In a Java EE Version 1.4 application, the identity assertion configuration is specified in the server-side extensions configuration and in the client-side bindings configuration.
- Migrate the Required Integrity and Required Confidentiality sections by configuring the Required Integrity and Required Confidentiality sections in an assembly tool. Migrating the Response Receiver

Configuration section is similar to migrating the Request Receiver Service Configuration Details section of the server-side extensions configuration. For more information, see “Migrating the JAX-RPC server-side extensions configuration” on page 93.

- Migrate the nonce configuration in the Login Config section in a Java EE Version 1.3 extensions configuration for Web Services Security to a Java EE Version 1.4 application.

  Important: Nonce is not configured in a Java EE Version 1.4 extension file for Web Services Security. Rather, it is configured in the binding file for Web Services Security.

  To configure a nonce in the binding file, define the com.ibm.wsspi.wssecurity.token.username.addNonce property in the token generator of the username token.
- Configure the Add Timestamp section under the Request Generator Configuration in the assembly tool if the **Add Created Time Stamp** option is configured in the Java EE Version 1.3 extensions.

Results

This set of steps describe the types of information that you need to migrate the client-side extensions configuration for Web Services Security for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the server-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the server-side bindings file.”

Migrating the server-side bindings file

You can migrate the server-side bindings configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mappings of the top-level sections under the server-side **Binding Configurations** tab from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 11. The mapping of the configuration sections. Use the binding configuration information for migration.

Java EE Version 1.3 Binding Configurations	Java EE Version 1.4 Binding Configurations
Request Receiver Binding Configuration Details	Request Consumer Service Binding Configuration Details
Response Sender Binding Configuration Details	Response Generator Binding Configuration Details

Consider the following steps to migrate the server-side bindings from Java EE Version 1.3 to Java EE Version 1.4. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

Procedure

- Migrate the configuration information under the Request Receiver Binding Configuration Details section of a Java EE Version 1.3 application.
  1. Migrate any trust anchor information that is specified in the Java EE Version 1.3 application to Java EE Version 1.4 using the Trust Anchor dialog.
  2. Migrate the information under the certificate store list that is specified in the Java EE Version 1.3 application to Java EE Version 1.4 by configuring the Certificate Store List section in the Java EE Version 1.4 application.
  3. Configure the key locator and token consumer information that is referenced from the Key Information dialog window. The configuration of the key locator and the token consumer depends upon the key information type. For example, if an X.509 certificate that is embedded in the `<wsse:Security>` security header is used for digital signature, complete the following steps:

a. For configuring the key locator, specify the `com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator` class as the key locator class and do not specify a key store.

b. For configuring the token consumer, select the `com.ibm.wsspi.wssecurity.token.509TokenConsumer` class, specify `X509 certificate token` for the value type Uniform Resource Identifier (URI), and specify `system.wssecurity.X509BST` in the jaas.config.name field. Also, you must specify the certificate path settings (the trust anchor reference and the certificate store reference) as part of the token consumer configuration.

4. Explicitly specify the key information type in the Key Information Dialog window. In a Java EE Version 1.3 application, the key information type, such as the security token reference and the key identifier, is not explicitly specified. The key information type is implied by the configuration. In a Java EE Version 1.4 application, you must specify the key information type explicitly using the Key Information Dialog when you have digital signature or encryption information in the binding file. Before you configure the key information, make sure that you have configured the key locator and token consumer information that is referenced from the Key Information dialog.

   When you configure the key information for either digital signature or encryption, you need to specify the correct key information type. The value of the key information type depends upon the type of mechanism that is used to reference the security token that is used for digitally signing or encrypting. The following information describes the Security token reference (or Direct reference) and the Key identifier, which are the most common, recommended key information types that are used for digitally signing and encrypting:

   Security token reference (or Direct reference)

   The security token is directly referenced using the Uniform Resource Identifiers (URIs). The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Reference URI="#mytoken" />
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

   Key identifier

   The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the `KeyIdentifier` value depends upon the token type. For example, a hash of the important elements of the security token is used for generating the `KeyIdentifier` value. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

   In the Key Information Dialog window, specify the names of the key locator and the token consumer that you configured previously. The Key name field is optional for the consumer side.

5. Migrate the information in the Signing Information section by configuring the Signing Information, Part References, and Transforms sections.
   - Specify the Signature method and Canonicalization method algorithms in the Signing Information Dialog window.
   - Specify the Digest method algorithm in the Part Reference Dialog window.

6. Migrate the information under the Encryption Information section. In the Encryption Information Dialog window, select the name of the Key Information element that is configured for encryption, and specify the `RequiredConfidentiality` part. Verify that the value for the selected `RequiredConfidentiality` part is the same name as the Required Confidentiality part that is configured in the extension file.

   The Login Mapping section in the Java EE Version 1.3 application maps to the Token Consumer configuration for the type of token that is specified by the authentication method. For example, to

migrate a Login Mappings configuration that uses the `BasicAuth` authentication method, configure a token consumer for the username token. To configure a token consumer for a username token, complete the following steps:

a. Select the `com.ibm.wsspi.wssecurity.UsernameTokenConsumer` token consumer class.
b. Specify the name of the Required Security Token configuration from the Extensions within in the **Security Token** field.
c. Select **Username Token** for value type.
d. Specify the `system.wssecurity.UsernameToken` value in the **jaas.config.name** field.

- Migrate the configuration information in the Response Sender Binding Configuration Details section of the Java EE Version 1.3 bindings file to the Response Generator Binding Configuration Details section of the Java EE Version 1.4 application. Configuring the Response Generator section is very similar to configuring the Request Consumer section.
  1. Migrate the information from the Key Locators section by using the Key Locator Dialog window in an assembly tool.
  2. Configure a token generator, which is referenced in the Key Information Dialog window. You must configure a token generator for every security token that is generated in the SOAP message. If the token generator is for an X.509 certificate that is used for digital signature or encryption, complete the following steps:
     a. For configuring the key locator, specify the `com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator` class as the key locator class and do not specify a key store.
     b. For configuring the token generator, select the `com.ibm.wsspi.wssecurity.X509TokenGenerator` class and specify `X509 certificate token` for the value type Uniform Resource Identifier (URI). The key store information that is specified for the token generator is the same information that is used for configuring the key locator. Therefore, the keystore information from the Key Locators configuration in a Java EE Version 1.3 application is used to configure the key locator and the token generator in a Java EE Version 1.4 application.
     c. In the Token Generator Dialog window, specify the key store information that is required by the callback handler to obtain the key information that is required for generating the token.
     d. For the callback handler, select the `com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler` class.
  3. Specify the names of the key locator and the token generator in the Key Information Dialog window that you configured previously. The Key name is required for the generator side. The key that is specified in the Key Information Dialog window must exist in the list of keys that is specified in the key locator configuration. Also, migrating the Signing Information and the Encryption Information configurations is similar to migrating the Signing Information and the Encryption Information configurations for the Request Receiver Binding Configuration section. Configuring the key information for the response generator section is similar to configuring the key information for the request consumer section.

Results

This set of steps describes the types of information that you need to migrate the server-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the client-side binding configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the client-side bindings file.”

Migrating the client-side bindings file

You can migrate the Web Services Security client-side binding configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mapping of the top-level sections under the client-side **Port Bindings** tab within a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 12. The mapping of the configuration sections. Use the binding configuration information for migration.

Java EE Version 1.3 binding configuration for Web Services Security	Java EE Version 1.4 binding configuration for Web Services Security
Security Request Sender Binding Configuration	Security Request Generator Binding Configuration
Security Response Receiver Binding Configuration	Security Response Consumer Binding Configuration

Consider the following steps to migrate the client-side binding configuration from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

Procedure

- Migrate the information in the Security Request Sender Binding Configuration section in a Java EE Version 1.3 application to a Java EE Version 1.4 application. The migrations process for the Security Request Sender Binding Configuration section is similar to the process for the Response Sender Binding Configuration Details section in the server-side binding configuration. For more information, see “Migrating the server-side bindings file” on page 96.
- Migrate the information in the Key Locators, Signing Information, and the Encryption Information sections of the Java EE Version 1.3 application to a Java EE Version 1.4 application. The migration process for these elements on the client side is similar to migration process on the server side. For more information, see “Migrating the server-side bindings file” on page 96.
- Migrate the information in the Login Bindings section in a Java EE Version 1.3 application to a Java EE Version 1.4 application. The migration of the Login Bindings section depends upon the value of the authentication method. If the authentication method is `BasicAuth` or `IDAssertion`, configure a token generator for the username token. If the authentication method is LTPA, select the `com.ibm.wsspi.wssecurity.token.LTPATokenGenerator` class as the token generator class. If the client-side bindings for the web service uses IDAssertion, complete the following steps:
  1. Configure a token generator for the authentication token of the original client.
  2. Define the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property and set its value to `true` in the Token Generator Dialog window within an assembly tool. If the original client is using a username token for authentication and if the target web service is using `BasicAuth` for authentication, configure the following token generators in the client-side binding file:
     - The username token of the original client. You must set the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property in the token generator of the original client.
     - The username token of the intermediary web service.
- Migrate the Security Response Receiver Binding Configuration section from a Java EE Version 1.3 application to a Java EE Version 1.4 application. Migrating the Security Response Receiver Binding Configuration section is similar to migrating the Request Receiver Binding Configuration Details section of the server-side bindings configuration. Migrate this information under the Security Response Consumer Binding Configuration section. For more information, see “Migrating the server-side bindings file” on page 96.

  To configure a nonce in the binding file, define the `com.ibm.wsspi.wssecurity.token.username.addNonce` property in the token generator of the username token.

Results

This set of steps describe the types of information that you need to migrate the Web Services Security client-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Verify that you have migrated both the server-side and the client-side extension and binding configurations for a Java EE Version 1.3 application to a Java EE Version 1.3 application. For more information, see “Migrating JAX-RPC Web Services Security applications to Version 8.0 applications” on page 91.

View web services client deployment descriptor

Use this page to view your client deployment descriptor.

This administrative console page applies only to Java API for XML-based RPC (JAX-RPC) applications.

Before you begin this task, the web services application must be installed.

By completing this task, you can gather information that enables your to maintain or configure binding information. After the web services application is installed, you can view the web services deployment descriptors.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules > *URI_name*** .
3. Under Web Services Properties, click **View web services client deployment descriptor extension**.

The information in the following implementation indicates how to configure your application-level bindings. If the web server is acting as a client, the default bindings are used. To configure the server-level bindings, which are the defaults, 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**.
3. To configure the cell-level bindings, click **Security > Web services**.

If you are using any of the following configurations, verify that the deployment descriptor is configured properly:

- Request signing
- Request encryption
- BasicAuth authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response digital signature verification
- Response decryption

Request signing

If the integrity constraints (digital signature) are specified, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security propertiess, click **Web Services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Signing information > New**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

Request encryption

If the confidentiality constraints (encryption) are specified, verify that you configured the encryption information in the binding files.

To configure the encryption parameters, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Encryption Information > New**.

To configure the key locators, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Additional properties, click **Web Services: Default bindings for Web Services Security > Key locators**.

BasicAuth authentication

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains the CallBack support implementations:

com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler
: This implementation prompts for basic authentication information, the user name and password, in an interface.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
: This implementation reads the basic authentication information from the binding file.

com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler
: This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login binding information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Identity (ID) Assertion authentication with BasicAuth TrustMode

Configure a login binding in the bindings file with a com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation. Specify a BasicAuth user name and password that a trusted ID evaluator on a downstream server trusts.

To configure the login binding information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Identity (ID) Assertion authentication with the Signature TrustMode

Configure the signing information in the bindings file with a signing key pointing to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure ID assertion, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Additional properties, click **JAX-WS and JAX-RPC security runtime > Login mappings > IDAssertion**.

To configure the login binding information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Response digital signature verification

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Signing information > New**.

To configure the trust anchors, 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**.
3. Under Additional properties, click **Trust anchors > New**.

To configure the collection certificate store, 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**.
3. Under Additional properties, click **Collection certificate store > New**.

Response decryption

If the confidentiality constraints (encryption) are specified, verify that you defined the encryption information.

To configure the encryption information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Encryption information > New**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

View web services server deployment descriptor

Use this page to view your server deployment descriptor settings.

This administrative console page applies only to Java API for XML-based RPC (JAX-RPC) applications.

Before you begin this task, the web services application must be installed.

By completing this task, you can gather information that enables you to maintain or configure binding information. After the web services application is installed, you can view the web services deployment descriptors.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Properties, click **View web services server deployment descriptor**.

WebSphere Application Server, Network Deployment has three levels of bindings: application-level, server-level, and cell-level. The information in the following implementation descriptions indicate how to configure your application-level bindings. To configure the server-level bindings, which are the defaults, 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**.

To configure the cell-level bindings, click **Security** > **JAX-WS and JAX-RPC security runtime**.

- Request digital signature verification
- Request decryption
- Basic authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response signing

- Response encryption

Request digital signature verification

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit > Signing information**.

To configure the trust anchor, 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**.
3. Under Additional properties, click **Trust anchors**.

To configure the collection certificate store, 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**.
3. Under Additional properties, click **Collection certificate store**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

Request decryption

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To configure the encryption information parameters, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit** > **Encryption information**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

Basic authentication

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains callback support implementations:

com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler
The implementation prompts for BasicAuth information (user name and password) in an interface panel.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This implementation reads the BasicAuth information from the binding file.

com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler
This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login mapping information, 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**.
3. Under Additional properties, click **Login mappings**.

Identity (ID) assertion authentication with the BasicAuth TrustMode

Configure a login binding in the bindings file with a `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler` implementation. Specify a user name and password for basic authentication that a TrustedIDEvaluator on a downstream server trusts.

To configure the login mapping information, 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**.
3. Under Additional properties, click **Login mappings**.

Identity (ID) assertion authentication with the signature TrustMode

Configure the signing information in the bindings file with a signing key that points to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure the login mapping information, 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**.

3. Under Additional properties, click **Login mappings**.

The Java Authentication and Authorization Service (JAAS) uses `WSLogin` as the name of the login configuration. To configure JAAS, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Authentication, click **Java Authentication and Authorization Service** > **Application logins**.

The value of the <TrustedIDEvaluatorRef> tag in the binding must match the value of the <TrustedIDEvaluator> name.

To configure the trusted ID evaluators, 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**.
3. Under Additional properties, click **Trusted ID evaluators**.

Response signing

If the integrity constraints (digital signature) are defined, verify that you have the signing information configured in the binding files.

To specify the signing information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Server security bindings**.
4. In the Request receiver binding column, click **Edit** > **Signing information**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

Response encryption

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To specify the encryption information, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Services Security properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit** > **Encryption information**.

To configure the key locators, 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**.
3. Under Additional properties, click **Key locators**.

Migrating the UDDI registry

With most scenarios, existing UDDI registries are migrated automatically when you migrate to the current level of WebSphere Application Server. However, if your existing UDDI registry uses a network Apache Derby database or a DB2 UDDI Version 2 database, there are some manual steps that you must take.

Before you begin

Migrate your installation of WebSphere Application Server. Ensure that you select the option to migrate applications, so that the UDDI registry application will be migrated.

About this task

If your existing UDDI registry uses an Oracle, embedded Apache Derby or DB2 UDDI Version 3 database, you do not have to complete any manual migration; the registry is migrated automatically when you migrate WebSphere Application Server and start the UDDI node for the first time after migration.

If your existing UDDI registry uses a network Apache Derby database or a DB2 UDDI Version 2 database, you must complete some manual steps to migrate the registry.

If the UDDI database uses Apache Derby Version 10.2, you must migrate the database. For details, see the topic in the related links.

Procedure

If your UDDI registry uses a network Apache Derby database, complete the following steps.

1. If you have a cluster that contains servers at different levels of WebSphere Application Server, ensure that any UDDI registries are running on servers that are at the current level of WebSphere Application Server. For example, if you have a cluster that spans two nodes, you can upgrade one node to the current level while the other node remains at a previous level, provided that any servers that are running a UDDI registry are at the current level.
2. Initialize the relevant UDDI node. The initialize process will complete some of the UDDI registry migration.
3. Enter the following commands as the database administrator, from *app_server_root*/derby/lib.

```
java -cp db2j.jar;db2jtools.jar com.ibm.db2j.tools.ij

connect 'jdbc:db2j:uddi_derby_database_path';

run 'app_server_root/UDDIReg/databaseScripts/uddi30crt_drop_triggers_derby.sql';

quit;

cd app_server_root/derby/migration

java -cp db2j.jar;db2jmigration.jar;../lib/derby.jar com.ibm.db2j.tools.MigrateFrom51
  jdbc:db2j:uddi_derby_database_path
```

   where

   - *uddi_derby_database_path* is the absolute path of the existing Apache Derby database, for example *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30
   - *app_server_root* is the root directory for the installation of WebSphere Application Server

Results

The UDDI database and data source are migrated, and the UDDI node is activated.

Note: When you migrate WebSphere Application Server, the post-upgrade log for the profile indicates that the migration of the UDDI database is partially complete, and is missing the steps for triggers, aliases, and stored statements. If you initially enabled the debug function, the debug log for the database indicates that there was a failure creating triggers. Ignore these messages; the UDDI node completes the migration of the database when the UDDI node starts. For more information about these log files, see the topic about verifying the Cloudscape automatic migration. Also refer to this topic if other errors appear in the logs.

If the migration of the UDDI database completes successfully, the following message appears in the server log:

```
CWUDQ0003I: UDDI registry migration has completed
```

If the following error appears, an unexpected error occurred during migration. The UDDI registry node is not activated. Check the error logs for the problem and, if you cannot solve it, refer to the problem determination information on the WebSphere Application Server support web page.

```
CWUDQ004W: UDDI registry not started due to migration errors
```

Setting up a UDDI migration data source

For DB2, you can set up a UDDI migration data source that you can use to reference a Version 2 UDDI registry database.

About this task

Only migration from DB2 is supported, so this procedure describes how to set up a DB2 data source.

Procedure

1. If a suitable JDBC Provider for DB2 does not already exist, create one. Select the following options:
   - Database type: DB2
   - Provider type: DB2 Universal JDBC Driver Provider
   - Implementation type: Connection pool data source
2. Create a data source for the Version 2 UDDI registry by following these steps:
   a. Click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the scope of the JDBC provider that you selected or created earlier. For example, select the following to show the JDBC providers at the server level:

```
Node=your_node_name, Server=your_server_name
```

   c. Select the JDBC provider that you created earlier.
   d. Select **[Additional Properties] Data sources**. Do not select **[Additional Properties] Data sources (WebSphere Application Server V4)**.
   e. Click **New** to create a new data source. The Create a data source wizard starts.
   f. In the first pane of the Create a data source wizard, enter the following data:

      Data source name
      Type a suitable name, such as `UDDI Datasource`.

      JNDI name
      Type `datasources/uddimigration` . This value is compulsory, and you must set it as it is shown.

   g. On the database-specific properties pane of the wizard, enter the following data:

Database name
Type `UDDI20`, or the name of the Version 2 UDDI DB2 database.

Use this data source in container managed persistence (CMP)
Ensure that the check box is cleared.

h. On the setup security aliases pane of the wizard, enter the following data:

Component-managed authentication alias
Select the alias for the DB2 user ID used to access UDDI Version 2 data, for example `MyNode/UDDIAlias`.

Mapping-configuration alias
Select `DefaultPrincipalMapping`. This field is in the Security settings section.

i. When the Create a data source wizard is finished, click the data source to display its properties, then add the following information:

Description
Type a suitable description of the data source.

Category
Type `uddi`.

Data store helper class name
Ensure that this is set to `DB2 Universal data store helper`.

3. Click **Apply**, then save the changes to the master configuration.
4. Test the connection to your UDDI database by selecting the check box next to the data source and clicking **Test connection**. A message similar to `Test Connection for datasource UDDI Datasource on server server1 at node MyNode was successful` is displayed. If you do not see this message, investigate the problem with the help of the error message.

What to do next

Continue with the migration, as described in the topic about migrating to Version 3 of the UDDI registry.

Migrating a UDDI database that uses Apache Derby

If a UDDI database that uses IBM Cloudscape or Apache Derby was created with WebSphere Application Server Version 6.1 or earlier and now uses Apache Derby Version 10.2 or later, you need to migrate the database. If you have a UDDI database that uses any other supported database, including versions of IBM Cloudscape or Apache Derby earlier than Apache Derby Version 10.2, you do not need to undertake this procedure.

Before you begin

Migrate your installation of WebSphere Application Server; ensure that you select the option to migrate applications, so that the UDDI registry application is migrated.

About this task

Use this procedure if a UDDI database currently uses Apache Derby Version 10.2 or later. In this version of the product, such databases are being used with Apache Derby Version 10.3. Typically, you need to migrate the database if a UDDI database that uses IBM Cloudscape or Apache Derby was created with WebSphere Application Server Version 6.1 or earlier, and you upgrade the servers to the current level of the product.

If you do not migrate the database, the following error occurs when you try to save a business entity in a UDDI registry that is running on the current level of the product, where the registry uses Apache Derby Version 10.2:

```
Serious technical error has occurred while processing the request.
```

Procedure

1. Ensure that any servers that use the UDDI database are stopped.
2. Use the following command to start the Apache Derby command prompt:

```
WAS_HOME/derby/bin/embedded/ij
```

3. Run the following commands at the command prompt. Substitute the UDDI database location in the CONNECT statement.

```
connect 'WAS_HOME/profiles/profileName/databases/com.ibm.uddi/UDDI30';

drop trigger ibmudi30.tr_upd_busallsvc_p;

create trigger ibmudi30.tr_upd_bservice_p
after update of businesskey on ibmudi30.bservice
referencing old as old_real_service
new as new_real_service
for each row mode db2sql update ibmudi30.busallservice
set ibmudi30.busallservice.owningbusinesskey = new_real_service.businesskey
where ibmudi30.busallservice.servicekey = new_real_service.servicekey
and ibmudi30.busallservice.owningbusinesskey != ibmudi30.busallservice.businesskey;

exit;
```

4. Restart the servers that use the UDDI database.

Appendix. Directory conventions

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

Default product locations - z/OS

app_server_root
: Refers to the top directory for a WebSphere Application Server node.

  The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

  The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
: Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

  The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

  The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
: Refers to the installation root directory for Web Server Plug-ins.

profile_root
: Refers to the home directory for a particular instantiated WebSphere Application Server profile.

  Corresponding product variables are *server.root* and *user.install.root.*

  In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
: Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

  The corresponding product variable is *smpe.install.root*.

  The default is /usr/lpp/zWebSphere/V8R0.

© IBM Corporation 2005, 2009

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

© Copyright IBM Corp. 2011

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.

© Copyright IBM Corp. 2011

Index

A

asynchronous beans
  interoperating 3

B

bindings
  migration
    clients 99
    servers 96

D

data sources
  migration 7
deployment descriptors
  web services
    clients 100
    servers 103
directory
  installation
    conventions 17, 111

E

Enterprise JavaBeans (EJB)
  migration 22

H

HTTP sessions
  migration 75

J

JAX-RPC
  application migration 91
  migration
    client-side extension 95
    server-side extension 93
JAX-WS
  security binding migration 91
JNDI
  bootstrap address 53

S

schedulers
  interoperation 57

W

web services
  migration 77, 86
web services security
  migration 90

© Copyright IBM Corp. 2011