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

Migrating WebSphere applications

Note

Before using this information, be sure to read the general information under “Notices” on page 93.

Compilation date: September 23, 2008

Contents

How to send your comments

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

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an 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.

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.

Chapter 1. Web applications

Migrating to Java Platform, Standard Edition (Java SE) 6

This product version supports the Java™ Platform, Standard Edition (Java SE) 6 specification. Its Java virtual machine provides a Java language compiler and runtime environment. Decide whether your new and existing applications will take advantage of the capabilities added by Java SE 6, adjust the just-in-time (JIT) mode if necessary, and begin the transition from deprecated functions.

About this task

The following JSRs are new in Java SE 6:

- JSR 105: XML Digital Signature Application Programming Interfaces (APIs)
- JSR 173: Streaming API for XML (StAX)
- JSR 181: Web Services Metadata
- JSR 199: Java Compiler API
- JSR 202: Java Class-File Specification Update
- JSR 221: Java DataBase Connectivity (JDBC) 4.0
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0
- JSR 223: Scripting for the Java Platform
- JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0
- JSR 250: Common Annotations
- JSR 269: Pluggable Annotation-Processing API

The new virtual machine specification adds several features and functions to benefit application developers, such as interfaces for integrating the Java and scripting languages, password prompting, file input-output enhancements, and parsing of streaming XML documents.

- Decide whether to take advantage of new Java SE 6 capabilities in your applications.

  You can deploy applications using Java SE 6 features only to Version 7 nodes, as earlier product versions do not provide the Java SE 6 virtual machine.

  Applications that access classes and APIs internal to the Java virtual machine might produce errors. These classes and APIs are not covered by the Java SE 6 specification and are therefore subject to change. Packages with prefixes such as com.sun.* are considered internal. Additionally, direct use of implementations of XML and XSL parsers is strongly discouraged, such as direct use of Xerces and Xalan classes that provide the Java API for XML Processing (JAXP) implementation for the virtual machine. The direct parser APIs also are considered internal and subject to change. Applications should rely only on the JAXP APIs defined in the Java SE 6 API documentation. If your application requires a specific version of Xerces or Xalan, or some other XML/XSL parser package, then embed the parser within your application's `WEB-INF/lib` directory and set the appropriate class loading mode in your application deployment so that for your application the XML parser APIs are loaded from the application class path, not the Java virtual machine bootstrap class path. Failure to follow this guideline can cause significant errors when you try to migrate to a new Java SE 6 level.
- Compile Java SE 6 applications to run on previous Java virtual machine levels by setting the compiler modes.

  When compiling applications that are built with Java SE 6 that are intended for running on previous specifications, specify `-source` and `-target` modes for the Java SE 6 compiler. Doing so ensures that the bytecode generated is compatible with the earlier Java virtual machine.

  For example, if the target Java virtual machine is at 1.4.2 level, when you compile applications with Java SE 6, you should specify `-source 1.4`, and `target 1.4` to generate bytecode compatible with 1.4.2. This does not handle the usage of packages, classes, or functions new to Java SE 6. It only

addresses bytecode output. Developers must take care in what APIs they are using from the J2SE packages if they intend to run the application on multiple Java virtual machine specification levels.

- Address incompatibilities in previously compiled Java 2 Standard Edition (J2SE) 1.4 and 5.0 applications.

  Java SE 6 is upwards binary-compatible with Java 2 Technology Edition, Version 5.0 and Java 2 Technology Edition, Version 1.4.2, except for the incompatibilities documented by Sun Microsystems at http://java.sun.com/javase/technologies/compatibility.jsp.

- Transition from deprecated Java Virtual Machine Debug Interface (JVMDI) and Java Virtual Machine Profiler Interface (JVMPI) functions to Java Virtual Machine Tool Interface (JVMTI).

  JVMDI and JVMPI functions were deprecated in J2SE 5.0. They have been removed from Java SE 6.

- Update your use of the Java command line interface.

  The command-line interfaces for the Java SE 6 level have not changed extensively from J2SE 5, although they vary among virtual machine vendors. You can find them in the `JAVA_HOME/bin` directory. Here are some notable command line options that are standard to all Java SE 6 implementations.

  – For JVMTI, use `-agentlib` to load a native agent library that you specify.
  – For JVMTI, use `-agentpath` to load the native agent library by the full path name.
  – For JVMTI, use `-javaagent` to load the Java programming language agent (see java.lang.instrument for details).
  – See `apt -help` for information about this new command line supporting the annotations capability.
  – See `javac -help` for information and updates to that command line.

- Update ANT tasks.

  If you have created ANT tasks based on the idltojava ANT task shipped with prior versions of this product, ensure that it passes the proper parameters for Java SE 6 as it does for J2SE 1.4 or 5, to ensure the stubs, ties and skeletons that it generates are compatible with earlier product releases.

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 %>, a will 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 will also break compatibility with some older JSP applications. However, as of JDK 1.4, importing classes from the unnamed package is not valid. See Java 2 Platform, Standard Edition Version 1.4.2 Compatibility with Previous Releases 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 prior to 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 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 on the basis of 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 things 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 uris 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 appeared as \$ in JSP 1.2 was \$, however, the output now is just $.

jsp:useBean tag

WebSphere® Application Server version 5.1 and later enforces more strict adherence to the specification for the jsp:useBean tag: with type and class attributes. Specifically, you should use the type attribute should be used to specify a Java type that cannot be instantiated as a JavaBean. 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 a JavaBean, the WebSphere Application Server JSP container produces a unrecoverable translation error at translation time.

Generated packages for JSP classes

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

JspServlet class

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

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

About this task

Migration of Web applications deployed in previous versions of WebSphere Application Server is usually not necessary. Version 2.2 and 2.3 of the Java Servlet specification and version 1.2 and 1.4 of the JavaServer Pages (JSP) specifications are still supported unless the behavior was changed in Servlet 2.4 or JSP 2.0 specifications. Version 2.4 of the Java Servlet specification and version 2.0 of the JSP specification are still supported unless the behavior was changed in that Servlet 2.5 or JSP 2.1 Specification. These changes are generally available in more detail in their corresponding specification.

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.

	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.x EAR files (deployed in Version 4.x 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.

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

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. Add the Web container custom property, com.ibm.ws.webcontainer.contenttypecompatibility, with a value of V4, V5, V6, V7. 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.

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 2. EJB applications

Migrating to Java Platform, Standard Edition (Java SE) 6

This product version supports the Java Platform, Standard Edition (Java SE) 6 specification. Its Java virtual machine provides a Java language compiler and runtime environment. Decide whether your new and existing applications will take advantage of the capabilities added by Java SE 6, adjust the just-in-time (JIT) mode if necessary, and begin the transition from deprecated functions.

About this task

The following JSRs are new in Java SE 6:

- JSR 105: XML Digital Signature Application Programming Interfaces (APIs)
- JSR 173: Streaming API for XML (StAX)
- JSR 181: Web Services Metadata
- JSR 199: Java Compiler API
- JSR 202: Java Class-File Specification Update
- JSR 221: Java DataBase Connectivity (JDBC) 4.0
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0
- JSR 223: Scripting for the Java Platform
- JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0
- JSR 250: Common Annotations
- JSR 269: Pluggable Annotation-Processing API

The new virtual machine specification adds several features and functions to benefit application developers, such as interfaces for integrating the Java and scripting languages, password prompting, file input-output enhancements, and parsing of streaming XML documents.

- Decide whether to take advantage of new Java SE 6 capabilities in your applications.

  You can deploy applications using Java SE 6 features only to Version 7 nodes, as earlier product versions do not provide the Java SE 6 virtual machine.

  Applications that access classes and APIs internal to the Java virtual machine might produce errors. These classes and APIs are not covered by the Java SE 6 specification and are therefore subject to change. Packages with prefixes such as com.sun.* are considered internal. Additionally, direct use of implementations of XML and XSL parsers is strongly discouraged, such as direct use of Xerces and Xalan classes that provide the Java API for XML Processing (JAXP) implementation for the virtual machine. The direct parser APIs also are considered internal and subject to change. Applications should rely only on the JAXP APIs defined in the Java SE 6 API documentation. If your application requires a specific version of Xerces or Xalan, or some other XML/XSL parser package, then embed the parser within your application's `WEB-INF/lib` directory and set the appropriate class loading mode in your application deployment so that for your application the XML parser APIs are loaded from the application class path, not the Java virtual machine bootstrap class path. Failure to follow this guideline can cause significant errors when you try to migrate to a new Java SE 6 level.
- Compile Java SE 6 applications to run on previous Java virtual machine levels by setting the compiler modes.

  When compiling applications that are built with Java SE 6 that are intended for running on previous specifications, specify `-source` and `-target` modes for the Java SE 6 compiler. Doing so ensures that the bytecode generated is compatible with the earlier Java virtual machine.

  For example, if the target Java virtual machine is at 1.4.2 level, when you compile applications with Java SE 6, you should specify `-source 1.4`, and `target 1.4` to generate bytecode compatible with 1.4.2. This does not handle the usage of packages, classes, or functions new to Java SE 6. It only

addresses bytecode output. Developers must take care in what APIs they are using from the J2SE packages if they intend to run the application on multiple Java virtual machine specification levels.

- Address incompatibilities in previously compiled Java 2 Standard Edition (J2SE) 1.4 and 5.0 applications.

  Java SE 6 is upwards binary-compatible with Java 2 Technology Edition, Version 5.0 and Java 2 Technology Edition, Version 1.4.2, except for the incompatibilities documented by Sun Microsystems at http://java.sun.com/javase/technologies/compatibility.jsp.

- Transition from deprecated Java Virtual Machine Debug Interface (JVMDI) and Java Virtual Machine Profiler Interface (JVMPI) functions to Java Virtual Machine Tool Interface (JVMTI).

  JVMDI and JVMPI functions were deprecated in J2SE 5.0. They have been removed from Java SE 6.

- Update your use of the Java command line interface.

  The command-line interfaces for the Java SE 6 level have not changed extensively from J2SE 5, although they vary among virtual machine vendors. You can find them in the `JAVA_HOME/bin` directory. Here are some notable command line options that are standard to all Java SE 6 implementations.

  – For JVMTI, use `-agentlib` to load a native agent library that you specify.
  – For JVMTI, use `-agentpath` to load the native agent library by the full path name.
  – For JVMTI, use `-javaagent` to load the Java programming language agent (see java.lang.instrument for details).
  – See `apt -help` for information about this new command line supporting the annotations capability.
  – See `javac -help` for information and updates to that command line.

- Update ANT tasks.

  If you have created ANT tasks based on the idltojava ANT task shipped with prior versions of this product, ensure that it passes the proper parameters for Java SE 6 as it does for J2SE 1.4 or 5, to ensure the stubs, ties and skeletons that it generates are compatible with earlier product releases.

Migrating enterprise bean code to the supported specification

Support for the Enterprise JavaBeans (EJB) 3.0 specification is added for this product.

Before you begin

There should not be migration issues associated with using EJB 3.0 beans. Existing applications should continue to run as-is and compile without error.

Note: The EJB 3.0 specification has deprecated the use of EJB 1.1 style entity beans. While using EJB 3.0 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.

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

     Note: The EJB Version 2.0 specification mandates that prior to the EJB container's running a findBy*Method* query, the state of all enterprise beans enlisted in the current transaction be synchronized with the persistent store. This is done so that the query is performed against current data. If 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.

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.

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.
   - 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.

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.

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());
   }
```

Container interoperability

Container interoperability describes the ability of the product clients and servers at different versions to successfully negotiate differences in native Enterprise JavaBeans (EJB) finder methods support and Java EE compliance.

Interoperability of the handle formats in WebSphere Application Server, Version 5 and Version 5.0.1

Applications that attempt to persist handles to enterprise beans and **EJBHome** needed to subclass ObjectInputStream in WebSphere Application Server, Version 5. This action was required so that the subclass ObjectInputStream could utilize the context class loader to resolve the classes for enterprise beans and EJBHome stubs.

In addition, handles created and persisted in WebSphere Application Server, Version 5 only work with objects that have an unchanged remote interface. If the remote interface is changed, the handle is no longer valid because the stub is serialized inside the handle and its serial Version UID changes if the remote interface changes.

This release introduces a new handle persistence mechanism that avoids the implementation drawbacks of the previous version. However, if handles are used for this WebSphere Application Server deployment, you should consider the following issues when applying this update, future WebSphere Application Server Fix Packs and EJB Container cumulative fixes for WebSphere Application Server, Version 5.

If a WebSphere Application Server, Version 5 persisted handle or home handle is encountered by a WebSphere Application Server, Version 5.0.1 system, it can be read and utilized. In addition, it will be converted to WebSphere Application Server, Version 5.0.1 format if it is re-persisted. The WebSphere Application Server, Version 5.0.1 format cannot be read by a WebSphere Application Server, Version 5 system unless PQ72184 is applied.

Problems arise when handles are persisted and shared across systems that are not at the WebSphere Application Server, Version 5.0.1 level or later. However, a Version 5 system can receive a handle from Version 5.0.1 remotely through a call to get a handle on an enterprise bean or a getHomeHandle on an **EJBHome**. The remote call will succeed, however, any attempt to persist it on the Version 5 system will have the same limitations regarding the use of ObjectInputStream and changes in remote interface invalidating the persisted handle.

When your application stores handles persistently and shares this persistence with multiple clients or application servers, apply WebSphere Application Server, Version 5.0.1 or PQ72184 to both the client and server systems at the same time. Failure to do so can result in the inability of these systems to read the handle data stored by upgraded systems. Also, handles stored by the WebSphere Application Server, Version 5 can force the applications of the updated system to still subclass `ObjectInputStream`. Applications using the WebSphere Application Server Enterprise, Version 5 scheduler and process choreographer, are affected by these changes. These users should update their Version 5 systems at the same time with either Version 5.0.1 or PQ72184.

If the applications store handles in the session context, or locally in a file on the same system, that is not shared by other applications, on different systems, they might be able to update their systems individually, rather than all at once. If Client Container and thin client applications do not share persisted handle data, they can be updated as needed as well. However, handles created and persisted in WebSphere Application Server, Version 5, Version 4.0.3 and later (with the property flag set), or Version 3.5.7 and later (with the property flag set) are not usable if either the home or the remote interface changes.

If any WebSphere Application Server, Version 3.5.7 or Version 4.0.3 and later enables the system property com.ibm.websphere.container.portable to **true**, any handles to objects on that server have the same interoperability limitations. In addition, if any WebSphere Application Server, Version 3.5.7 and later or Version 4.0.3 applications store a handle obtained from a WebSphere Application Server, Version 5 or Version 5.0.1, the same restrictions apply, regarding the need to subclass ObjectInputStream and the usability of handles after a change to the remote interface is made.

Replication of the Http Session and Handles

This note applies to you if you place Handles to Homes or Enterprise JavaBeans, or EJB or EJBHome references in the Http Session in your application and you use Http Session Replication. If you intend to replicate a mixed environment of Version 5.0.0 and Version 5.0.1 or 5.0.2 machines you should first apply the latest Version 5.0.0 container cumulative e-fix to the Version 5.0.0 machines before allowing the Version 5.0.1 or 5.0.2 server into the typology. The reason for this is that Version 5.0.0 servers are not able to understand the persisted Handle format used on the Version 5.0.1 and 5.0.2 server. This is similar

to the case of Version 5.0.0 and Version 5.0.1 or 5.0.2 systems trying to use a shared database, mentioned above. But in this case, it is the Http Session object and not the database providing the persistence.

Top Down Deployment Mapping

The size of the Handle objects has grown due to the fix put in to allow serialization and deserialization to occur without the previous requirements of subclassing the ObjectInputStream and so on. Top down deployment of an object that contains EJB and EJBHome references create a database table ddl that has a field of 1000 bytes of VARCHAR for BITDATA which will contain the Handle. It might be that your object's Handle does not fit in the 1000 byte default field, and you might need to adjust this to a higher value. You might try increments of 250 bytes, that is, 1250, 1500, and so on.

Chapter 3. Client applications

clientUpgrade script

The clientUpgrade script migrates application client modules and their resources in an enterprise archive (EAR) file so that these application clients can run in WebSphere Application Server Version 7. The script converts an EAR file that you want to migrate and then overwrites the original EAR file with the converted EAR file.

The following title provides information about the clientUpgrade script.

Note: This command was deprecated in Version 6.1.

Type	Description
Product	The clientUpgrade script is available in the WebSphere Application Server (Express and Base) product only.
Authority	To run this script, your user profile must have *ALLOBJ authority.

Syntax	The syntax of the clientUpgrade script is:
	`clientUpgrade` *`EAR_file`* `[ -clientJAR` *`client_JAR_file`* `]`
	`[ -logFileLocation` *`logFileLocation`* `]`
	`[ -traceString` *`trace_spec`* `[ -traceFile` *`file_name`* `] ]`

Parameters	The parameters of the clientUpgrade script are: • *EAR_file* -- This is a required parameter. The value EAR_file specifies the fully-qualified path of the EAR file that contains the application client modules that you want to migrate. • -clientJAR -- This is an optional parameter. The value client_JAR_file specifies a JAR file that you want to migrate. The script overwrites the original EAR file with a new EAR file that contains only the specified JAR files. If you do not specify this parameter, the clientUpgrade script migrates all client JAR files in the EAR file. • -logFileLocation -- Use this optional parameter to specify an alternate location to store the log output. • -traceString -- This is an optional parameter. The value trace_spec specifies the trace information that you want to collect. To gather all trace information, specify ″*=all=enabled″ (including the double quotation marks (″)). By default, the script does not gather trace information. If you specify this parameter, you must also specify the -traceFile parameter. • -traceFile -- This is an optional parameter. The value file_name The value file_name specifies the name of the output file for trace information. If you specify the -traceString parameter but do not specify the -traceFile parameter, the script does not generate a trace file.
Logging	The clientUpgrade script displays status while it runs. It also saves more extensive logging information to the clientupgrade.log file. This file is located in the /QIBM/UserData/WebSphere/AppServer/V7/edition/profiles/default/logs directory (for a default installation using the default profile) or in the location specified by the -logFileLocation parameter.

These examples demonstrate correct syntax. In this example, the My51Application.ear file is migrated from WebSphere Application Server Version 5.1, The script overwrites the original EAR file with a new file that you can deploy in your WebSphere Application Server Version 7 profile.

```
clientUpgrade /My51Application/My51Application.ear
```

In this example, only the `myJarFile.jar` client JAR file is migrated. The script overwrites `My51Application.ear` with an EAR file that contains `myJarFile.jar`. You can deploy the new EAR file in your WebSphere Application Server profile.

```
clientUpgrade /My51Application/My51Application.ear -clientJAR myJarFile.jar
```

Chapter 4. Web services

Web Services-Interoperability Basic Profile

The Web Services-Interoperability (WS-I) Basic Profile is a set of non-proprietary Web services specifications that promote interoperability. WebSphere Application Server conforms to the WS-I Basic Profile Version 1.1 and WS-I Basic Security Profile Version 1.0.

The WS-I Basic Profile is governed by a consortium of industry-leading corporations, including IBM, under direction of the WS-I Organization. The profile consists of a set of principles that relate to bringing about open standards for Web services technology. All organizations that are interested in promoting interoperability among Web services are encouraged to become members of the Web Services Interoperability Organization.

Several technology components are used in the composition and implementation of Web services, including messaging, description, discovery, and security. Each of these components are supported by specifications and standards, including SOAP 1.1, Extensible Markup Language (XML) 1.0, HTTP 1.1, Web Services Description Language (WSDL) 1.1, and Universal Description, Discovery and Integration (UDDI). The WS-I Basic Profile specifies how these technology components are used together to achieve interoperability, and mandates specific use of each of the technologies when appropriate. You can read more about the WS-I Basic Profile at the WS-I Organization Web site.

As technology components are updated, these components are also used in the composition and implementation of Web Services. One example is that both SOAP 1.1 and SOAP 1.2 are now supported.

Note: Building on the support for WS-I Basic Profile Version 1.0, WS-I Basic Profile V1.1, Attachment Profile V1.0, and Basic Security Profile (BSP) V1.0, you can implement Web services with WebSphere Application Server Version 7.0 using the following current emerging standard WS-I profiles:

- *WS-I Basic Profile V1.2* builds on WS-I Basic Profile V1.0 and WS-I Basic Profile V1.1 and adds support for WS-Addressing (WS-A) and SOAP Message Transmission Optimization Mechanism (MTOM). The WS-Addressing specification enables the asynchronous message exchange pattern so that you can decouple the service request from the service response. The SOAP header of the sender's request contains the wsa:ReplyTo value that defines the endpoint reference to which the provider's response is sent. Decoupling the request from the response enables long running Web services interactions. Leveraging the asynchronous programming model support in JAX-WS Version 2.1 in combination with WS-Addressing, you can now take advantage of the ability to create Web services invocations where the client can continue to process work without waiting for a response to return. This provides for a more dynamic and efficient model to invoke Web services. Using MTOM, you can send and receive binary data optimally within a SOAP message.
- *WS-I Basic Profile V2.0* builds on top of Basic Profile V1.2 with the addition of support for SOAP 1.2.
- *WS-I Basic Security Profile V1.1* extends the WS-I Basic Security Profile V1.0 standard by profiling the latest WS-Security V1.1 specification.
- *WS-I Reliable Secure Profile 1.0* builds on WS-I Basic Profile V1.2, WS-I Basic Profile V2.0, WS-I Basic Security Profile V1.0, and WS-I Basic Security Profile V1.1 and adds support for WS-Reliable Messaging 1.1, WS-Make Connection 1.0, and WS-Secure Conversation 1.3. WS-Reliable Messaging 1.1 is a session-based protocol that provides message level reliability for Web services interactions. WS-Make Connection 1.0 was developed by the WS-Reliable Messaging workgroup to address scenarios where a Web services endpoint is behind a firewall or the endpoint has no visible endpoint reference. If a Web services endpoint loses connectivity during a reliable session, WS-Make Connection provides an efficient method to re-establish the reliable session. Additionally, WS-Secure Conversation V1.3 is a session-based security protocol

that uses an efficient symmetric key based encryption algorithm for message level security. WS-I Reliable Secure Profile V1.0 provides secure reliable session-oriented Web services interactions.

Each of the technology components has requirements that you can read about in more detail at the WS-I Organization Web site. For example, support for Universal Transformation Format (UTF)-16 encoding is required by WS-I Basic Profile. UTF-16 is a kind of Unicode encoding scheme that uses 16-bit values to store Universal Character Set (UCS) characters. UTF-8 is the most common encoding that is used on the Internet; UTF-16 encoding is typically used for Java and Windows® product applications; and UTF-32 is used by various Linux® and UNIX® systems. Unlike UTF-8, UTF-16 has issues with big-endian and little-endian, and often involves Byte Order Mark (BOM) to indicate the endian. BOM is mandatory for UTF-16 encoding and it can be used in UTF-8.

The application server only supports UTF-8 and UTF-16 encoding of SOAP messages.

The following table summarizes some of the properties of each UTF:

Bytes	Encoding form
EF BB BF	UTF-8
FF FE	UTF-16, little-endian
FE FF	UTF-16, big-endian
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian

BOM is written prior to the XML text, and it indicates to the parser how the XML is encoded. The XML declaration contains the encoding, for example: `<?xml version=xxx encoding="utf-xxx"?>`. BOM is used with the encoding to determine how to interpret the XML. Here is an example of a SOAP message and how BOM and UTF encoding are used:

```
POST http://www.whitemesa.net/soap12/add-test-rpc HTTP/1.1
Content-Type: application/soap+xml; charset=utf-16; action=""
SOAPAction:
Host: localhost: 8080
Content-Length: 562

0xFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
 xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
 xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
 xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <q1:echoString xmlns:q1="http://soapinterop.org/">
   <inputString soap:encodingStyle="http://example.org/unknownEncoding"
   xsi:type="xsd:string">
    Hello SOAP 1.2
   </inputString>
  </q1:echoString>
 </soap:Body>
</soap:Envelope>
```

In the example code, `0xFF0xFE` represents the byte codes, while the *<?xml>* declaration is the textual representation.

Support for `styleEncoding` is not supported in SOAP 1.2 so here is the same example of the SOAP message but without the encoding information:

```
OxFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
   xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
```

```
 xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
 xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <q1:echoString xmlns:q1="http://soapinterop.org/">
   <inputString xsi:type="xsd:string">
    Hello SOAP 1.2
   </inputString>
  </q1:echoString>
 </soap:Body>
</soap:Envelope>
```

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

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.

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 have been deprecated. 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, see 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. For more information, 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, read about securing Web services for Version 6 applications based on WS-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.

Migrating Apache SOAP Web services

See Migrating Apache SOAP Web services to Web Services for J2EE standards to learn how to migrate Apache SOAP Web services. This topic explains how to 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.

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

To migrate these client applications according to Java standards:

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 Gallery. 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.

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 the UDDI registry

With most scenarios, migration of existing UDDI registries happens 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 need to perform any manual migration; migration happens 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 your UDDI registry uses a DB2 UDDI Version 2 database, follow the steps in “Migrating to Version 3 of the UDDI registry” on page 25 and sub-topics.
- 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 WebSphere Application Server Version 7.0. For example, if you have a cluster that spans two nodes, you can upgrade one node to WebSphere Application Server Version 7.0 while the other node remains at a previous level, provided that any servers that are running a UDDI registry are at Version 7.0.
  2. Initialize the relevant UDDI node. The initialize process will perform 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 “Verifying the Cloudscape automatic migration” on page 34. 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 IBM software support.

```
CWUDQ004W: UDDI registry not started due to migration errors
```

Migrating to Version 3 of the UDDI registry

Use this topic to migrate a Version 2 UDDI registry that uses a DB2 database, running in WebSphere Application Server Version 5.1.x to a Version 3 UDDI registry running in WebSphere Application Server Version 7.0. WebSphere Application Server Version 6.x already includes the Version 3 UDDI registry, so if you are migrating from Version 6.x, you do not need to perform this procedure.

Before you begin

The following constraints apply to this procedure:

- Your existing registry must use a DB2 database.
- Your existing registry must run in WebSphere Application Server Version 5.1.x.

1. Stop the UDDI registry application that is running in your Version 5.1.x application server. This prevents further UDDI requests being directed to the UDDI registry and ensures that no new data is published during the migration process.
2. Record information about the uddi.properties values being used. This file is located in the *DeploymentManager_install_dir*/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name* directory on your WebSphere Application Server Version 5.1.x system (or in the properties subdirectory if you are migrating a standalone application server).
3. Migrate the server from WebSphere Application Server Version 5.1.x to Version 7.0. This results in a new directory tree for the migrated Version 7.0 application server.
4. Start the new, migrated, Version 7.0 application server.
5. Create a new data source for the Version 2 UDDI database. This data source is known as the *UDDI migration data source*. The JNDI name must be `datasources/uddimigration`. To complete this step, see Setting up a UDDI migration datasource.
6. Set up the UDDI Version 3 registry and migrate the Version 2 data.

   Follow the instructions in the topic in the information center on setting up a customized UDDI node, including the subtopic that relates to node initialization. The topic describes how to perform the following actions:

   - Create the Version 3 DB2 database.
   - Create the J2C authentication data entry.
   - Create the JDBC provider and data source.
   - Deploy the UDDI registry application.
   - Start the server.
   - Configure and initialize the node. The UDDI registry node initialization detects the UDDI migration data source, and migrates the Version 2 data as part of the UDDI node initialization processing. This data migration can take some time, depending on the amount of data in your UDDI registry.

Results

The UDDI registry is migrated. If the UDDI node remains in a state of initialization pending, migration pending or value set creation pending, check the server log for errors. If the following message appears, an unexpected error occurred during migration. Check the error logs for the problem and, if it cannot be fixed, see the IBM software support Web site at http://www.ibm.com/software/support.

```
CWUDQ004W: UDDI registry not started due to migration errors
```

After the problem is fixed, you can complete the migration by clicking **Initialize** again.

Verify that the migration process completed successfully by checking for the following message in the server log:

```
CWUDQ0003I: UDDI registry migration has completed
```

What to do next

After migration is complete, you can remove the UDDI migration data source, and the registry is available for use.

Setting up a UDDI migration data source

Use this topic to set up a UDDI migration data source, to be used to reference a Version 2 UDDI registry database.

About this task

Migration is only supported from DB2, so these instructions describe how to set up a DB2 data source.

1. If a suitable JDBC Provider for DB2 does not already exist, then create one, selecting the options DB2 Universal JDBC Driver Provider and Connection Pool data source.

   For details on how to create a JDBC provider, see the article in the information center on creating and configuring a JDBC provider using the administrative console.
2. Create a data source for the Version 2 UDDI registry by following these steps:
   a. Click **Resources** → **JDBC** → **JDBC Providers**.
   b. Select the desired scope of the JDBC provider you selected or created earlier. For example, select:

```
Server: yourservername
```

      to show the JDBC providers at the server level.
   c. Select the JDBC provider created earlier.
   d. Under **Additional Properties**, select **Data sources** (*not* the **Data sources (WebSphere Application Server V4)** option).
   e. Click **New** to create a new data source.
   f. In the **Create a data source** wizard, enter the following data:

      Name a suitable name, such as UDDI Datasource

      JNDI name
      set to **datasources/uddimigration** - this value is compulsory, and must be as shown.

      Component-managed authentication alias
      select the alias for the DB2 userid used to access UDDI Version 2 data, for example MyNode/UDDIAlias
   g. Click **Next**.
   h. On the database specific properties page of the wizard, enter the following data:

      Database name
      `UDDI20`, or the name given to your Version 2 UDDI DB2 database.

Use this Data Source in container-managed persistence (CMP)
ensure the check box is cleared.

i. Click **Next**, then check the summary and click **Finish**.

j. Click the data source to display its properties, and add the following information:

Description
a suitable description

Category
set to **uddi**

Data store helper class name
filled in for you as: com.ibm.websphere.rsadapter.DB2DataStoreHelper

Mapping-configuration alias
set to DefaultPrincipalMapping

3. Click **Apply** and 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**. You will see a message similar to ″Test Connection for datasource UDDI Datasource on server server1 at node MyNode was successful″. 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 detailed in Migrating to Version 3 of the UDDI registry.

Chapter 5. Service integration

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.x to WebSphere Application Server Version 7.0. 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.x node to a Version 7.0 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

1. Launch a scripting command. For more information, refer to Starting the wsadmin scripting client.
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, you may need to create a new data store table.

About this task

When migrating from older versions of WebSphere Application Server to Version 7.0, 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 using the sibDDLGenerator tool located in the bin directory of your WebSphere Application Server installation.

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

Chapter 6. Data access resources

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.

- 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.
     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 the 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.

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 “Upgrading Cloudscape manually” on page 37

37 topic. 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.

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>
   ```

   Note: 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

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. Consult “Upgrading Cloudscape manually” for instructions.

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.

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.

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.
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.

Chapter 7. Messaging resources

Migrating from WebSphere Application Server Version 5 embedded messaging

This set of topics describe the migration of JMS applications from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in WebSphere Application Server Version 7.0.

About this task

You can temporarily use JMS resources developed for WebSphere Application Server Version 5.1 with service integration in WebSphere Application Server Version 7.0, however you should ideally migrate all resources to Version 7.0 as soon as possible.

For general migration considerations, see "General considerations for migrating from Version 5 embedded messaging."

When migrating a Version 5.1 node to Version 7.0, 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.

Note: It is intended that WebSphere Application Server Version 5.1 will no longer support connections from application servers or Java EE application clients to the JMS server component of embedded messaging. When this happens, it will no longer be possible to define JMS resources for the Version 5.1 default messaging provider. It is also intended that WebSphere Application Server Version 5.1 will no longer support connections from client applications executing in a Version 5.1 environment, or applications using Version 5.1 default messaging provider resources, to the messaging engines in WebSphere Application Server Version 7.0. When this happens, it will no longer be possible to define a WebSphere MQ client link property for messaging engines.

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 52
- "Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 1" on page 54
- "Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2" on page 58
- "General considerations for migrating from Version 5 embedded messaging"
- "Migrating Version 5.1 messages using the message migration utility" on page 45

General considerations for migrating from Version 5 embedded messaging

This topic describes the general considerations for migrating from WebSphere Application Server Version 5 embedded messaging to the Version 7.0 default messaging provider. These general considerations apply to the related migration tasks.

- "You do not need to change Version 5.1 JMS applications" on page 42
- "You do not need to change Version 5.1 JMS resource definitions" on page 42
- "You do not need to change Version 5.1 client JMS resource definitions" on page 43

You do not need to change Version 5.1 JMS applications

When migrating a WebSphere Application Server Version 5.1 node to Version 7.0, 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 need 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 need to change Version 5.1 JMS resource definitions

When migrating a WebSphere Application Server Version 5.1 node to Version 7.0, 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's 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 Version 7.0 default messaging provider.

If a node is migrated to WebSphere Application Server Version 7.0, the JMS resources are not changed, except to use the Version 7.0 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 WebSphere Application Server Version 7.0 whenever the associated Version 5 default messaging resources are used.

After migration, the JMS resources are implemented through 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. For example, in the WebSphere Application Server Version 7.0 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 Version 7.0 default messaging provider JMS resources as soon as is conveniently possible (after all JMS applications using those resources have been moved onto WebSphere Application Server Version 7.0). This enables you to benefit from the better performance of the Version 7.0 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 Version 7.0 JMS resources with values appropriate to your Version 7.0 environment and your use of the Version 5.1 properties.

New JMS resources should be created as Version 7.0 JMS resources. Any wsadmin or JMX scripts that create Version 5.1 JMS resources need to be changed to create Version 7.0 JMS resources for use in WebSphere Application Server Version 7.0.

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

JMS client applications that have JMS resources configured using the Application Client Resource Configuration Tool (ACRCT) in WebSphere Application Server Version 5.1 should continue to work without change with WebSphere Application Server Version 7.0.

You should replace Version 5.1 default messaging JMS resources with equivalent Version 7.0 default messaging provider JMS resources as soon as is conveniently possible (after all the application servers have been migrated onto WebSphere Application Server Version 7.0). You should use the Version 7.0 ACRCT to replace the JMS resources and to create any new JMS resources. New JMS resources should be created as Version 7.0 JMS resources.

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

When migrating a WebSphere Application Server Version 5.1 node to Version 7.0, any messages (and knowledge of durable subscriptions) held by the JMS server are not migrated automatically. Whether you need 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 and stopping 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 Version 7.0 node, to access the Version 5.1 JMS server, get the Version 5.1 messages, and resend or publish them to applications on WebSphere Application Server Version 7.0.
- 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 45.

  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 WebSphere Application Server Version 7. 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 Version 7.0 default messaging provider is implemented.

The Version 7.0 default messaging provider 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 Version 7.0 default messaging provider.

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 Version 7.0 default messaging provider. This should not present a problem because the Version 7.0 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 need to be changed, any wsadmin or JMX scripts that make use of the listener port retry count need to be changed to make use of the “Maximum failed deliveries” setting for use in WebSphere Application Server Version 7.0.

Use of Version 7.0 bus resources 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 WebSphere Application Server Version 7.0 default messaging provider 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 WebSphere Application Server Version 7 protocols used by messaging engines. This link can be used to access JMS resources backed by a destination anywhere on the bus or on any other connected bus.

If you migrate a WebSphere Application Server Version 5.1 node to Version 7.0, 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

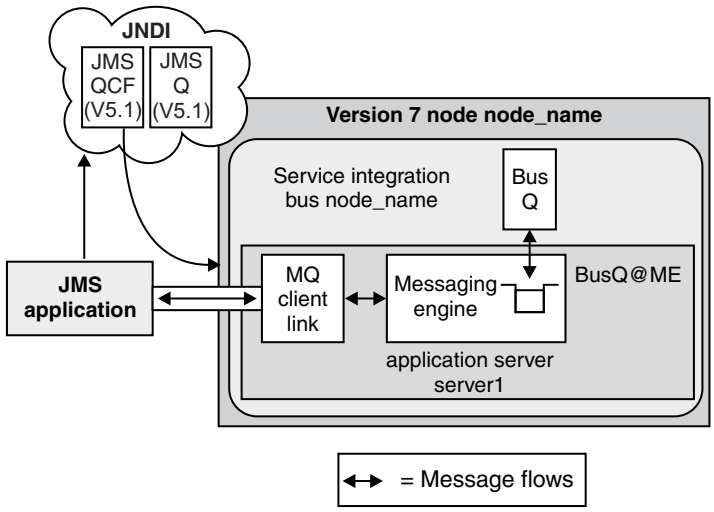

Figure 1. WebSphere Application Server 5 JMS application scenario after migration. This figure shows an example single-node scenario after migrating the node to WebSphere Application Server Version 7.0. The JMS resources are now managed as Version 5 embedded messaging JMS resources implemented by the 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 Version 7.0

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 a Version 7.0 administrative interface 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 to Version 7.0, the Version 5 embedded messaging configuration scripts are retained, but have no influence on the WebSphere Application Server Version 7.0 default messaging provider and fail if run. **Do not run the scripts.** Failure of the scripts can appear to be a failure of the WebSphere Application Server Version 7.0 installation, but no damage or action to the WebSphere Application Server 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 Version 7.0 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 WebSphere Application Server Version 7.0 default messaging queues.

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

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

WebSphere Application Server message migration utility

This topic describes the 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 wish 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 Version 7.0 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: Version 7.0 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 WebSphere Application Server Version 7.0.

About this task

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

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 WebSphere Application Server Version 7.0 installableApps directory.
   c. Accept all the defaults on the next screen and click **Next**.
   d. You do not need 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

This topic describes how to run the WebSphere Application Server message migration utility.

Before you begin

1. Ensure that you have a WebSphere Application Server Version 5.1 system that contains an embedded messaging server.
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 you have a running WebSphere Application Server Version 7.0 system that fulfills the following conditions.
   a. The WebSphere Application Server Version 7.0 system is on the same host as the WebSphere Application Server Version 5.1 system.
   b. The WebSphere Application Server Version 7.0 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 Version 7.0).

   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 Version 7.0 node.

About this task

Messages are migrated from the WebSphere Application Server Version 5.1 system to the Version 7.0 system under an XA (globally coordinated) transaction. For further information about the XA transaction, see “XA recovery” on page 48.

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

During successful migration, messages are moved from the WebSphere Application Server Version 5.1 system to the WebSphere Application Server Version 7.0 system. No copy is left on the Version 5.1 server queue.

Note:

1. You run the message migration utility only once for a particular queue, unless a failure occurs.
2. 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 retry a failing message migration, or delete the message, the message ordering of the remaining messages is preserved on the WebSphere Application Server Version 7.0 queue.

1. Follow the actions indicated by the message migration utility.

2. On the panel called ″Select the direction of migration″ select **Migrate messages from Version 5 to Version 7**.
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. You need to run the migration utility more than once if the first message migration attempt fails.
2. If a queue has a failing message then perform one of the following operations:
   a. Retry the failing message, 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.

Reversing the migration of messages using the message migration utility

This topic describes how to reverse the results of the WebSphere Application Server message migration utility.

Before you begin

If you want to revert to using WebSphere Application Server Version 5.1, instead of Version 7.0, you must reverse the migration of the messages before deleting the Version 7.0 server.

About this task

Reverse migration is carried out under XA coordination.

If the reverse migration takes place after a partial migration from WebSphere Application Server Version 5.1 to Version 7.0, 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.

Note:

1. You run the message migration utility only once for a particular queue, unless a failure occurs.
2. 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 retry a failing message migration, or delete the message, the message ordering of the remaining messages is preserved on the WebSphere Application Server Version 7.0 queue.

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. Retrying the failing message, 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

This topic describes the XA transaction and how it relates to the process of migrating messages.

Before you begin

For XA recovery to complete successfully:

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

   Note: This is not a problem for the WebSphere Application Server Version 7.0 default messaging provider (the service integration bus), 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 doubt, information about the resources is saved in the transaction log.

About this task

During migration, messages are removed from the WebSphere Application Server Version 5 embedded messaging and put into the service integration bus under an XA transaction. 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. 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 doubt″. 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. If a transaction is in doubt, the transaction manager attempts to recover the transaction when the application server next restarts. 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 Version 7.0 server. For example, if the profile in use is ″Profile1″, then the log file is located in the following directory:

user_data_root/profiles/Profile1/logs/message_migration_utility.log

, where *user_data_root* is the user information 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 7.0
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 7.0
   V6 reliability    : ReliablePersistent
```

1. Start the Version 5 embedded messaging queue manager.
2. Start the Version 7.0 application server, if it is not already started.

Results

XA recovery continues automatically.

Migration of message fields

This topic shows the message changes that can occur when using the WebSphere Application Server message migration utility.

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 which **can** change as a result of migration:

Table 2. JMS Message Header fields

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.

Table 2. JMS Message Header fields (continued)

Header field name	State after migration
JMSType	Unchanged
JMSDestination	The name of the destination is unaltered. Other properties of the destination are mapped to their WebSphere Application Server Version 7 equivalents, where possible.
JMSReplyTo	The name of the reply destination is unaltered, and assumed to exist on the WebSphere Application Server Version 7.0 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 WebSphere Application Server Version 7.0 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

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

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

Table 4. JMS_IBM properties (continued)

JMS_IBM property name	State after migration
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 WebSphere Application Server Version 7.0 default messaging provider.

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 and stopping 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 Version 7.0, you do not need to make any changes to JMS applications which 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 “General considerations for migrating from Version 5 embedded messaging” on page 41.

Before migrating, consider the stand-alone application server scenario shown in the following figure Figure 2 on page 53.

- The JMS application uses JNDI to look up the JMS resources in the WebSphere Application Server name space.
- 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.

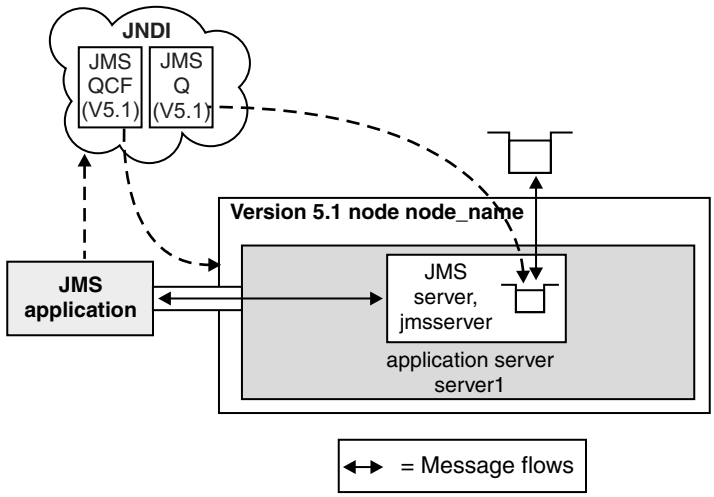

Figure 2. Stand-alone WebSphere Application Server 5 JMS application scenario before migration. This figure shows the example stand-alone application server scenario before migrating the stand-alone application server to WebSphere Application Server Version 7.0. The JMS application is supported by an application server. The JMS application could 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 Version 7.0 default messaging provider, complete the following steps:

1. Migrate the stand-alone WebSphere Application Server to Version 7.0. See the documentation about migrating product configurations. 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 Version 7.0 default messaging provider. For example, after migrating the application server, use the Version 7.0 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 the application server, the basic stand-alone application server scenario becomes as shown in the following figure Figure 3 on page 54.

- 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.
- 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 WebSphere Application Server Version 7.0 protocols used by messaging engines.

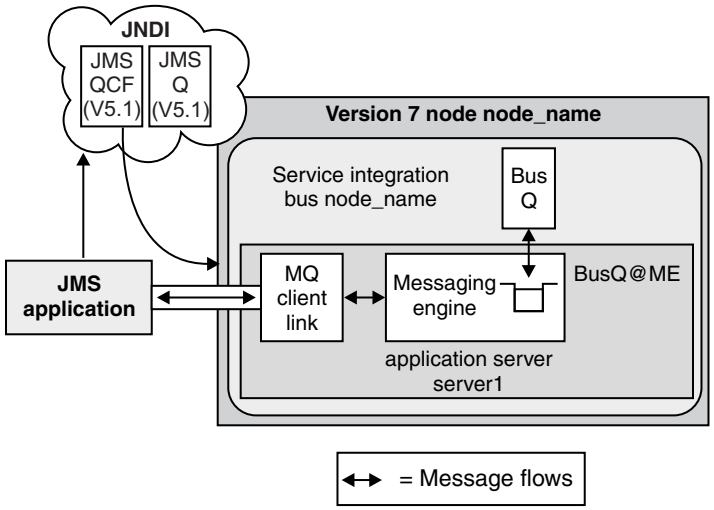

Figure 3. WebSphere Application Server 5 JMS application scenario after migration. This figure shows an example stand-alone application server scenario after migrating the application server to WebSphere Application Server Version 7.0. The JMS resources are now managed as Version 5 default messaging 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

Note: If you have configured authorization level security on Version 5.1 it cannot be migrated to Version 7.0. 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 Version 7.0 default messaging provider JMS resources as soon as is conveniently possible (after all JMS applications using those resources have been moved onto the later version of WebSphere Application Server).

You should define any new JMS resources as Version 7.0 resources; 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

This topic provides an example of the migration of a message-driven bean application from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in WebSphere Application Server Version 7.0.

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 and stopping 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 Version 7.0, 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.

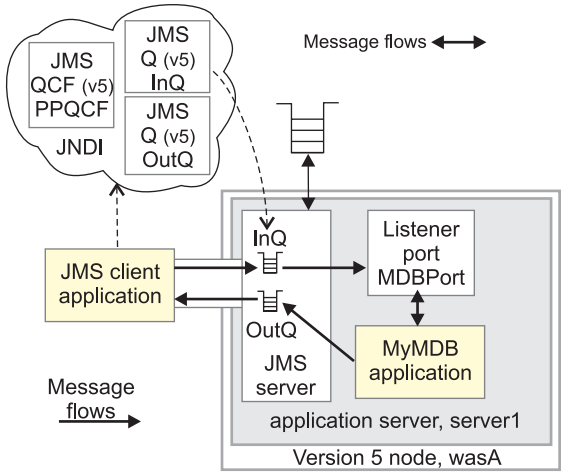

Figure 4. WebSphere Application Server Version 5.1 single-node MDB application scenario before migration

- 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.

  WebSphere Queue, InQ

Name:	InQ
JNDI Name:	jms/SampleInputQueue

  All other properties have default settings.

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” and “Version 7.0” is used to refer to “WebSphere Application Server Version 7.0”. For example, “Version 5.1 JMS resources” refers to JMS resources provided by WebSphere Application Server Version 5.1.

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 name space. 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 5 on page 57.

- 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 Version 7.0 default messaging provider JMS resources as soon as is conveniently possible (for example, after any Version 5.1 JMS client applications have been migrated onto WebSphere Application Server Version 7.0).

To migrate the MDB application from Version 5 embedded messaging to the Version 7.0 default messaging provider, complete the following steps:

1. Migrate the WebSphere Application Server node to Version 7.0. See the documentation about migrating product configurations. 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's 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 Version 7.0 default messaging provider.

Results

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

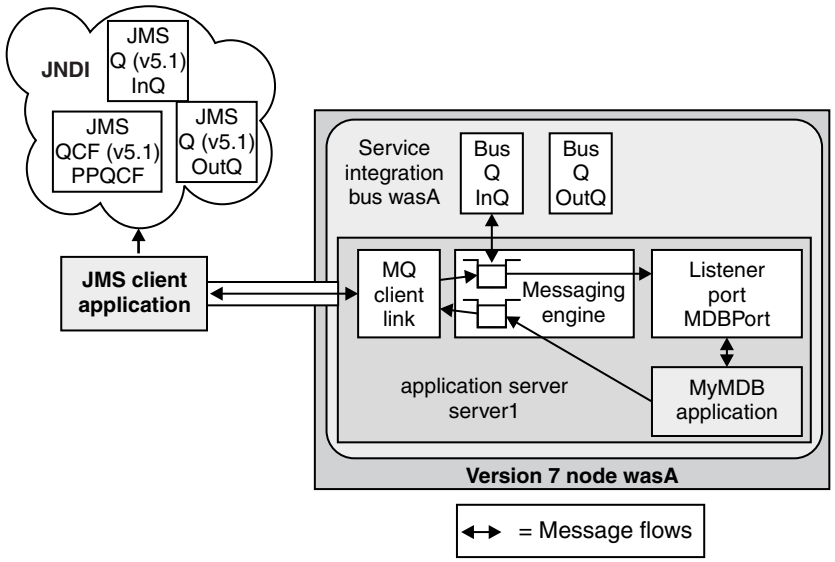

Figure 5. WebSphere Application Server Version 5.1 node after migration

The MDB application can continue to access the JMS resources, which are now implemented through the WebSphere Application Server 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.

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 Version 7.0 default messaging provider JMS resources as soon as is conveniently possible (after all JMS applications using those resources have been moved onto WebSphere Application Server Version 7.0).
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 Version 7.0 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 could replace the resources by writing some scripting that retrieves the Version 5.1 property values then creates Version 7.0 JMS resources with values appropriate to your Version 7.0 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 Version 7.0 default messaging provider JMS resources including JMS activation specification, see “Example: Migrating a message-driven bean from Version 5 embedded messaging - stage 2” on page 58.

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 7.0 node from using Version 5.1 default messaging JMS resources and listener port to using Version 7.0 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 54.

Consider the example scenario, before replacing the Version 5.1 JMS resources with Version 7.0 JMS resources, shown in the following figure Figure 6.

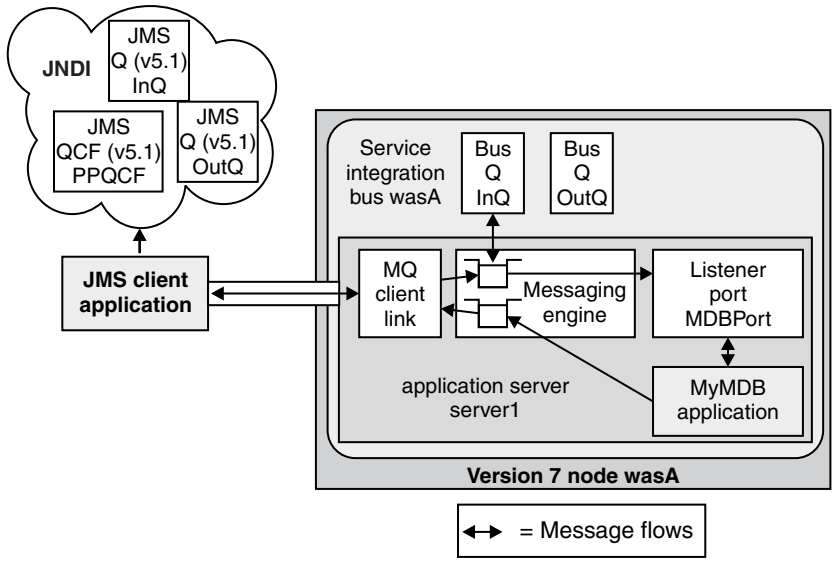

Figure 6. MDB application scenario before replacing the Version 5 JMS resources with Version 7.0 JMS resources. The migrated scenario still uses the Version 5.1 JMS resources: a JMS queue connection factory, PPQCF; two JMS queues, InQ and OutQ; and a listener port, MDBPort. These resources are to be replaced by equivalent Version 7.0 resources.

- 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 Version 7 WebSphere Application Server administrative console the queue connection factory can be found by clicking **Resources** → **JMS** → **JMS providers** → **V5 default messaging provider** → **[Additional Properties] Queue connection factories**.
- On the Version 7.0 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 Version 7.0 default messaging provider JMS resources, complete the following steps:

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 Version 7.0 JMS queue connection factory 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 also need to rewrite the MDB application to use JMS 1.1 interfaces). For example, use the Version 7.0 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 Version 7.0 JMS queues to replace the Version 5.1 JMS queues For example, use the Version 7.0 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 Version 7.0 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 Version 7.0 equivalents, the MDB application scenario becomes as shown in the following figure:

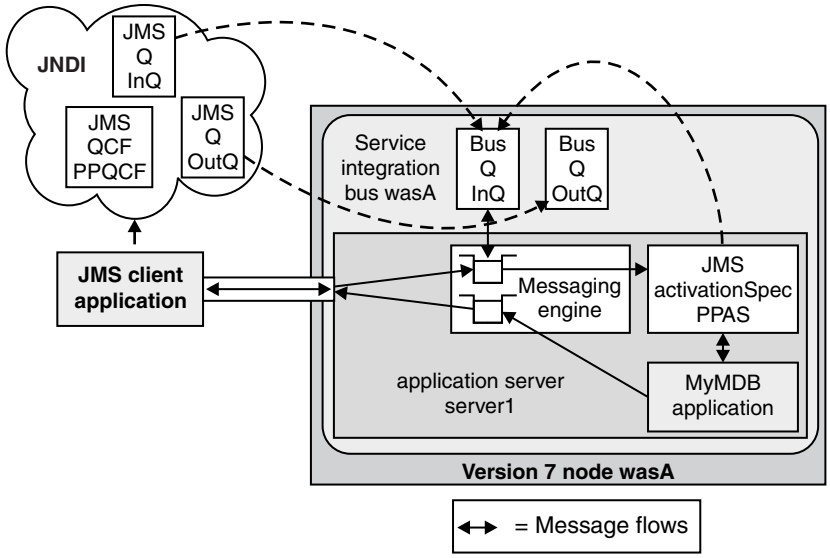

Figure 7. MDB application scenario after replacing the Version 5.1 JMS resources with Version 7.0 JMS resources. The migrated scenario still uses the Version 5.1 JMS resources, but equivalent Version 7.0 resources have been created for use by the MDB application when it has been redeployed as a JCA 1.5-compliant resource with the JMS activation specification. The Version 7.0 resources are: a JMS queue connection factory, PPQCF; two JMS queues, InQ and OutQ; and a JMS activation specification, PPAS.

6. Redeploy the MDB application to use the JMS activation specification, as described in Deploying and managing 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.
10.

Results

You should now be able to use the MDB application with the Version 7.0 JMS resources. 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.

Chapter 8. Security

Migrating, coexisting, and interoperating – Security considerations

Use this topic to migrate the security configuration of previous WebSphere Application Server releases and its applications to the new installation of WebSphere Application Server.

Before you begin

This information addresses the need to migrate your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server 7.0 or later. Complete the following steps to migrate your security configurations:

- If security is enabled in the previous release, obtain the administrative server ID and password of the previous release. This information is needed in order to run certain migration jobs.
- You can optionally disable security in the previous release before migrating the installation. No logon is required during the installation.

Follow the steps in ″Migrating product configurations″.

Results

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 7.0.

What to do next

If a custom user registry is used in the previous version, the migration process does not migrate the class files that are used by the standalone custom registry in the previous *app_server_root*/classes directory. Therefore, after migration, copy your custom user registry implementation classes to the *app_server_root*/classes directory.

If you upgrade from WebSphere Application Server, Version 5.*x* to WebSphere Application Server Version 7.0, the data that is associated with Version 5.x trust associations is not automatically migrated. To migrate trust associations, see “Migrating trust association interceptors” on page 68.

If the previous version instance is configured to enable secure connections using digital certificates that are signed by the Digital Certificate Manager (DCM) local certificate authority, those certificates must be renewed. For example, they must be renewed for the previous version instance, the WebSphere Application Server Version 7.0 profile, and all of the Secure Socket Layer-enabled clients and servers that connect to WebSphere Application Server. For more information, see SSL handshake failure using digital certificates signed by a Digital Certificate Manager (DCM) local certificate authority.

i5/OS® *SYSTEM certificate stores for applications are deprecated in WebSphere Application Server Version 5. In WebSphere Application Server Version 7.0, you must migrate your applications to use Java keystores.

The os400.security.password.validation.list.object property is profile-dependent. If you are migrating from Version 5, see “Migrating Java thin clients that use the password encoding algorithm” on page 77 for instructions on how to migrate your client configuration.

Interoperating with previous product versions

IBM WebSphere Application Server inter-operates with the previous product versions. Use this topic to configure this behavior.

Before you begin

The current release of the Application Server distinguishes the identities of the user who acts as an administrator, managing the Application Server environment, from the identity of the user that is used for authenticating between servers. In prior releases, the end user had to specify a server user ID and password as the user identity for authenticating between servers. In the current release of the Application Server, the server user ID is generated automatically and internally; however, the end user can specify that the server user ID and password not be automatically generated. This option is especially important in the case of a mixed-release cell, where the server user ID and password are specified in a down-level version of the Application Server. In such a scenario, the end user should opt out of automatically generating the server user ID and instead use the server user ID and password that is specified in the down-level version of the Application Server, in order to ensure backwards compatibility.

Interoperability is achieved only when the Lightweight Third Party Authentication (LTPA) authentication mechanism and a distributed user registry is used such as Lightweight Directory Access Protocol (LDAP) or a distributed Custom user registry. LocalOS on most platforms is not considered a distributed user registry (except on z/OS® within the z/OS environment). Also, the Simple WebSphere Authentication Mechanism (SWAM) cannot be used for interoperability as it does not contain credentials that can be forwarded outside of the existing process.

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

1. Configure WebSphere Application Server Version 7.0 with the same distributed user registry (that is, LDAP or Custom) that is configured with the previous version. Make sure that the same LDAP user registry is shared by all of the product versions.
   a. In the administrative console, select **Security > Global security**.
   b. Choose an available Realm definition and click **Configure**.
   c. Enter a **Primary administrative user name**. This identity is the user with administrative privileges that is defined in your local operating system. If you are not using the local OS ad the user registry, select the **Server identity that is stored in the user repository**, enter the Server user ID, and the associated password. The user name is used to log on to the administrative console when administrative security is enabled. WebSphere Application Server Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

      Note: In WebSphere Application Server, Versions 5.x and 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 7.0, this identity is used as the server user identity. You need to specify another user for the administrative user identity.
   d. When interoperating with Version 6.0.x or previous versions, you must select the Server identity that is stored in the user repository. Enter the **Server user id** and the associated **Password.**
2. Configure the LTPA authentication mechanism. Automatic generation of the LTPA keys should be disabled. If not, keys used by a previous release are lost. Export the current LTPA keys from WebSphere Application Server Version 7.0 and import them into the previous release.
   a. In the administrative console select **Security > Global security**.
   b. From Authentication mechanisms and expiration, click **LTPA**.
   c. Click the **Key set groups** link , then click the key set group that displays in the Key set groups panel.
   d. Clear the **Automatically generate keys** check box.
   e. Click **OK**, then click **Authentication mechanisms and expiration** in the path at the top of the Key set groups panel.
   f. Scroll down to the Cross-cell single sign-on section, and enter a password to use for encrypting the LTPA keys when adding them to the file.

g. Enter the password again to confirm the password.
h. Enter the **Fully qualified key file name** that contains the exported keys.
i. Click **Export keys**.
j. Follow the instructions provided in the previous release to import the exported LTPA keys into that configuration.

3. If you are using the default SSL configuration, extract all of the signer certificates from the WebSphere Application Server Version 7.0 common trust store. Otherwise, extract signers where necessary to import them into the previous release.
   a. In the administrative console, click **Security > SSL certificate and key management**.
   b. Click **Key stores and certificates**.
   c. Click **NodeDefaultTrustStore**.
   d. Click **Signer certificates**.
   e. Select one signer and click **Extract**.
   f. Enter a unique path and filename for the signer. For example, `/tmp/signer1.arm`.
   g. Click **OK**. Repeat for all of the signers in the trust store.
   h. Check other trust stores for other signers that might need to be shared with the other server. Repeat steps e through h to extract the other signers.

   You can also import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring. the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server. For more information, see
4. Add the exported signers to `DummyServerTrustFile.jks` and `DummyClientTrustFile.jks` in the `/etc` directory of the back-level product version. If the previous release is not using the dummy certificate, the signer certificate(s) from the previous release must be extracted and added into the WebSphere Application Server Version 7.0 release to enable SSL connectivity in both directions.
   a. Open the key management utility, iKeyman, for that product version.
   b. Start `ikeyman.bat` or `ikeyman.sh` from the `${USER_INSTALL_ROOT}/bin` directory.
   c. Select **Key Database File > Open**.
   d. Open `${USER_INSTALL_ROOT}/etc/DummyServerTrustFile.jks`.
   e. Enter WebAS for the password.
   f. Select **Add** and enter one of the files extracted in step 2. Continue until you have added all of the signers.
   g. Repeat steps c through f for the `DummyClientTrustFile.jks` file.
5. Verify that the application uses the correct Java Naming and Directory Interface (JNDI) name and naming bootstrap port for performing a naming lookup.
6. Stop and restart all of the servers.

Migrating custom user registries

If you built your own custom user registry, consider the migration items listed below. If you have a custom user registry that was provided by a Security Solution Provider, you must contact that provider to ensure that you have the correct version of their custom user registry to support WebSphere Application Server.

Before you begin

In WebSphere Application Server, in addition to the UserRegistry interface, the custom user registry requires the Result object to handle user and group information. This file is already provided in the package and you are expected to use it for the getUsers, getGroups, and the getUsersForGroup methods.

You cannot use other WebSphere Application Server components, for example, data sources, to initialize the custom registry because other components, like the containers, are initialized after security and are not

available during the registry initialization. A custom registry implementation is a pure custom implementation, independent of other WebSphere Application Server components.

The getCallerPrincipal enterprise bean method and the getUserPrincipal and getRemoteUser servlet methods return the security name instead of the display name. For more information, see the API documentation.

If the migration tool is used to migrate the WebSphere Application Server Version 5 configuration to WebSphere Application Server Version 6.0.x and later, this migration does not change your existing code. Because the WebSphere Application Server Version 5 custom registry works in WebSphere Application Server Version 6.0.*x* and later without any changes to the implementation, except when using data sources, you can use the Version 5-based custom registry after the migration without modifying the code.

In WebSphere Application Server Version 6.0.*x* and later, a case-insensitive authorization can occur when using an enabled custom user registry.

Setting this flag does not have any effect on the user names or passwords. Only the unique IDs that are returned from the registry are changed to lower-case before comparing them with the information in the authorization table, which is also converted to lowercase during runtime.

Before proceeding, look at the UserRegistry interface. See Developing standalone custom registries for a description of each of these methods in detail.

About this task

The following steps go through all the changes that are required to move your WebSphere Application Server Version 4.*x* custom user registry that implemented the old com.ibm.websphere.security.CustomRegistry interface to the com.ibm.websphere.security.UserRegistry interface.

Note: The sample implementation file is used as an **example** when describing the following steps.

1. Change your implementation to `UserRegistry` instead of `CustomRegistry`. Change:

   ```
   public class FileRegistrySample implements CustomRegistry
   ```

   to:

   ```
   public class FileRegistrySample implements UserRegistry
   ```

2. Create the java.rmi.RemoteException exception in the constructors:

   ```
   public FileRegistrySample() throws java.rmi.RemoteException
   ```

3. Change the mapCertificate method to take a certificate chain instead of a single certificate. Change

   ```
   public String mapCertificate(X509Certificate cert)
   ```

   to:

   ```
   public String mapCertificate(X509Certificate[]cert)
   ```

   Having a certificate chain gives you the flexibility to act on the chain instead of one certificate. If you are interested only in the first certificate, take the first certificate in the chain before processing. In WebSphere Application Server Version 6.0.*x* and later, the mapCertificate method is called to map the user in a certificate to a valid user in the registry when certificates are used for authentication by the Web or the Java clients.

4. Remove the getUsers method.
5. Change the signature of the getUsers(String) method to return a Result object and accept an additional parameter (int). Change:

   ```
   public List getUsers(String pattern)
   ```

to:

```
public Result getUsers(String pattern, int limit)
```

In your implementation, construct the Result object from the list of the users that is obtained from the user registry (whose number is limited to the value of the limit parameter) and call the setHasMore method on the Result object if the total number of users in the registry exceeds the limit value.

6. Change the signature of the getUsersForGroup(String) method to return a Result object and accept an additional parameter (int) and throw a new exception called NotImplementedException exception. Change the following code:

   ```
   public List getUsersForGroup(String groupName)
             throws CustomRegistryException,
                    EntryNotFoundException {
   ```

   to:

   ```
   public Result getUsersForGroup(String groupSecurityName, int limit)
             throws NotImplementedException,
                    EntryNotFoundException,
                    CustomRegistryException {
   ```

   In WebSphere Application Server Version 6.0.x and later, this method is not called directly by the WebSphere Application Server security component. However, other components of WebSphere Application Server, like the WebSphere Business Integration Server Foundation process choreographer, use this method when staff assignments are modeled using groups. Because this implementation is supported in WebSphere Application Server Version 6.0.x and later, it is recommended that you change the implementation similar to the getUsers method as explained in step 5.

7. Remove the getUniqueUserIds(String) method.
8. Remove the getGroups method.
9. Change the signature of the getGroups(String) method to return a Result object and accept an additional parameter (int). Change the following code:

   ```
   public List getGroups(String pattern)
   ```

   to:

   ```
   public Result getGroups(String pattern, int limit)
   ```

   In your implementation, construct the Result object from the list of the groups that is obtained from the user registry whose number is limited to the value of the limit parameter. Call the setHasMore method on the Result object if the total number of groups in the registry exceeds the limit value.

10. Add the createCredential method. This method is not called at this time, so return as `null`.

    ```
    public com.ibm.websphere.security.cred.WSCredential
       createCredential(String userSecurityName)
             throws CustomRegistryException,
                    NotImplementedException,
                    EntryNotFoundException {
             return null;
       }
    ```

    The first and second lines of the previous code example are split onto two lines for illustrative purposes only.

11. To build the WebSphere Application Server Version 6.0.*x* and later implementation, make sure you have the `com.ibm.ws.runtime.jar` file in your class path.

    To set the files in your class path, use the following code as a sample and substitute your environment values for the variables that are used in the example:

    ```
    javac -J-Djava.version=1.6 -classpath
    app_server_root/plugins/com.ibm.ws.runtime.jar  FileRegistrySample.java
    ```

    Type the previous lines as one continuous line.

To build the WebSphere Application Server Version 5 custom registry (CustomRegistry) in WebSphere Application Server Version 6.0.x and later, only the `com.ibm.ws.runtime.jar` file is required.

12. Copy the implementation classes to the product class path.

    It is recommended that you copy these implementation classes to the *profile_root*`/classes` directory.

13. Use the administrative console to set up the custom registry.

    Follow the instructions in Configuring standalone custom registries to set up the custom registry, including the **Ignore case for authorization** option. Make sure that you add the WAS_UseDisplayName properties if required.

Results

WebSphere Application Server Version 4.*x* based custom user registry that implemented the old com.ibm.websphere.security.CustomRegistry interface is migrated to the com.ibm.websphere.security.UserRegistry interface.

What to do next

If you are enabling security, see Enabling security to complete the remaining steps. When completed, save the configuration and restart all the servers. Try accessing some Java Platform, Enterprise Edition (Java EE) resources to verify that the custom registry migration is successful.

Migrating trust association interceptors

Use this topic to manually migrate trust associations.

Before you begin

Note: Data sources are not supported for use within a Trust Association Interceptor (TAI). Data sources are intended for use within J2EE applications and designed to operate within the EJB and Web containers. Trust Association Interceptors do not run within a container, and while data sources may function in the TAI environment, they are untested and not guaranteed to function properly.

The following topics are addressed in this document:

- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

Changes to the product-provided trust association interceptors

For the product-provided implementation for the WebSEAL server, a new optional com.ibm.websphere.security.webseal.ignoreProxy property is added. If this property is set to `true` or `yes`, the implementation does not check for the proxy host names and the proxy ports to match any of the host names and ports that are listed in the com.ibm.websphere.security.webseal.hostnames and the com.ibm.websphere.security.webseal.ports property respectively. For example, if the VIA header contains the following information:

```
HTTP/1.1 Fred (Proxy), 1.1 Sam (Apache/1.1),
HTTP/1.1 webseal1:7002, 1.1 webseal2:7001
```

and the com.ibm.websphere.security.webseal.ignoreProxy property is set to `true` or `yes`, the host name `Fred`, is not used when matching the host names. By default, this property is not set, which implies that any proxy host names and ports that are expected in the VIA header are listed in the host names and the ports properties to satisfy the isTargetInterceptor method.

The previous VIA header information was split onto two lines for illustrative purposes only.

For more information about the com.ibm.websphere.security.webseal.ignoreProxy property, see the article in the information center on configuring single signon using trust association interceptor ++.

Migrating product-provided trust association interceptors

The properties that are located in the `webseal.properties` and `trustedserver.properties` files are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.*x* using the trust association panels in the administrative console. For more information, see Configuring trust association interceptors.

Changes to the custom trust association interceptors

If the custom interceptor extends the com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor property, implement the following new method to initialize the interceptor:

```
public int init (java.util.Properties props);
```

WebSphere Application Server checks the return status before using the trust association implementation. Zero (0) is the default value for indicating that the interceptor is successfully initialized.

However, if a previous implementation of the trust association interceptor returns a different error status, you can either change your implementation to match the expectations or make one of the following changes:

Method 1:

Add the com.ibm.websphere.security.trustassociation.initStatus property in the trust association interceptor custom properties. Set the property to the value that indicates the interceptor is successfully initialized. All of the other possible values imply failure. In case of failure, the corresponding trust association interceptor is not used.

Method 2:

Add the com.ibm.websphere.security.trustassociation.ignoreInitStatus property in the trust association interceptor custom properties. Set the value of this property to `true`, which tells WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

The public int init (java.util.Properties props method replaces the public int init (String propsFile) method.

The init(Properties) method accepts a java.util.Properties object, which contains the set of properties that is required to initialize the interceptor. All of the properties set for an interceptor are sent to this method. The interceptor can then use these properties to initialize itself. For example, in the product-provided implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to come from trusted hosts and ports. A return value of Zero (0) implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is not used.

The init(String) method still works if you want to use it instead of implementing the init(Properties) method. The only requirement is that you enter the file name containing the custom trust association properties using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using either of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server.

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating `.config` to the

com.ibm.websphere.security.trustassociation.types property value. If the myTAI.properties file is located in the *profile_root*/properties directory, set the following properties:

- com.ibm.websphere.security.trustassociation.types = myTAItype
- com.ibm.websphere.security.trustassociation.myTAItype.config = *profile_root*/properties/myTAI.properties

Method 2:

You can set the com.ibm.websphere.security.trustassociation.initPropsFile property in the trust association custom properties to the location of the file. For example, set the following property:

```
com.ibm.websphere.security.trustassociation.initPropsFile=
profile_root/properties/myTAI.properties
```

The previous line of code is split into two lines for illustrative purposes only. Type as one continuous line.

However, it is highly recommended that your implementation be changed to implement the init(Properties) method instead of relying on the init (String propsfile) method.

Migrating custom trust association interceptors

The trust associations from previous versions of WebSphere Application Server are not automatically migrated to WebSphere Application Server Version 6.0.*x* and later. You can manually migrate these trust associations using the following steps:

1. Recompile the implementation file, if necessary.

   For more information, refer to the ″Changes to the custom trust association interceptors″ section previously discussed in this document.

   a. Enter QSH from a command line to start the QShell environment.
   b. Change to the directory that contains your Java source file.
   c. Enter the command to recompile the implementation file.

   ```
   javac -Djava.version=1.6 -classpath
   app_server_root/plugins/com.ibm.ws.runtime.jar:install_root/lib/j2ee.jar your_implementation_file.java
   ```

2. Copy the custom trust association interceptor class files to a location in your product class path. Copy these class files into the *profile_root*/classes directory.
3. Start WebSphere Application Server.
4. Enable security to use the trust association interceptor. The properties that are located in your custom trust association properties file and in the trustedserver.properties file are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.*x* and later using the trust association panels in the administrative console.

   For more information, see Configuring trust association interceptors.

Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)

Use this topic as an example of how to perform programmatic login using the CORBA-based programmatic login APIs.

Before you begin

This document outlines the deprecated Common Object Request Broker Architecture (CORBA) programmatic login APIs and the alternatives that are provided by JAAS. WebSphere Application Server fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login application programming interfaces (API). Refer to the *Securing applications and their environment* PDF for more details on JAAS support.

The following list includes the deprecated CORBA programmatic login APIs.
- `profile_root/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but it is not recommended that you use the API.

The APIs that are provided in WebSphere Application Server are a combination of standard JAAS APIs and a product implementation of standard JAAS interfaces.

The following information is only a summary; refer to the JAAS documentation for your platform located at: http://www.ibm.com/developerworks/java/jdk/security/ .
- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:
    com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
    Provides a non-prompt CallbackHandler handler when the application pushes basic authentication data (user ID, password, and security realm) or token data to product login modules. This API is recommended for server-side login.
    com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl
    Provides a login prompt CallbackHandler handler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

    If this API is used on the server side, the server is blocked for input.
  - javax.security.auth.callback.Callback interface:
    javax.security.auth.callback.NameCallback
    Provided by JAAS to pass the user name to the LoginModules interface.
    javax.security.auth.callback.PasswordCallback
    Provided by JAAS to pass the password to the LoginModules interface.
    com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl
    Provided by the product to perform a token-based login. With this API, an application can pass a token-byte array to the LoginModules interface.
  - **javax.security.auth.spi.LoginModule interface**

    WebSphere Application Server provides a LoginModules implementation for client and server-side login. Refer to the *Securing applications and their environment* PDF for details.
- javax.security.Subject:
  com.ibm.websphere.security.auth.WSSubject
  An extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject
  com.ibm.websphere.security.cred.WSCredential
  After a successful JAAS login with the WebSphere Application Server LoginModules interfaces, a com.ibm.websphere.security.cred.WSCredential credential is created and stored in the Subject.
  com.ibm.websphere.security.auth.WSPrincipal
  An authenticated principal that is created and stored in a Subject that is authenticated by the WebSphere Application Server LoginModules interface.

1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs: The CORBA-based programmatic login APIs are replaced by JAAS login.

   Note: The LoginHelper application programming interface (API) that is used in the following example is deprecated in WebSphere Application Server Version 7.0 and will be removed in a future release. It is recommended that you use the JAAS programmatic login APIs that are shown in the next step.

```
public class TestClient {
...
private void performLogin() {
// Get the ID and password of the user.
```

```
String userid = customGetUserid();
String password = customGetPassword();

// Create a new security context to hold authentication data.
LoginHelper loginHelper = new LoginHelper();
try {
// Provide the ID and password of the user for authentication.
org.omg.SecurityLevel2.Credentials credentials =
loginHelper.login(userid, password);

// Use the new credentials for all future invocations.
loginHelper.setInvocationCredentials(credentials);
// Retrieve the name of the user from the credentials
// so we can tell the user that login succeeded.

String username = loginHelper.getUserName(credentials);
System.out.println("Security context set for user: "+username);
} catch (org.omg.SecurityLevel2.LoginFailed e) {
// Handle the LoginFailed exception.
}
}
...
}
```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login APIs.

   The following example assumes that the application code is granted for the required Java 2 security permissions. For more information, see the *Securing applications and their environment* PDF and the JAAS documentation located at http://www.ibm.com/developerworks/java/jdk/security/.

```
public class TestClient {
...
private void performLogin() {
// Create a new JAAS LoginContext.
javax.security.auth.login.LoginContext lc = null;

try {
// Use GUI prompt to gather the BasicAuth data.
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by  login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
```

```
);

// Retrieve the name of the principal from the Subject
// so we can tell the user that login succeeded,
// should only be one WSPrincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPrincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPrincipal p =
(com.ibm.websphere.security.auth.WSPrincipal) it.next();
System.out.println("Principal: " + p.getName());
}
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
}
...
}
```

Migrating from the CustomLoginServlet class to servlet filters

Use this topic to allow migration in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

Before you begin

The CustomLoginServlet class is deprecated in WebSphere Application Server Version 5. Those applications using the CustomLoginServlet class to perform authentication now need to use form-based login. Using the form-based login mechanism, you can control the look and feel of the login screen. In form-based login, a login page is specified and displays when retrieving the user ID and password information. You also can specify an error page that displays when authentication fails.

If login and error pages are not enough to implement the CustomLoginServlet class, use servlet filters. Servlet filters can dynamically intercept requests and responses to transform or use the information that is contained in the requests or responses. One or more servlet filters attach to a servlet or a group of servlets. Servlet filters also can attach to JavaServer Pages (JSP) files and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both form-based login and servlet filters are supported by any Servlet 2.3 specification-compliant Web container. A form login servlet performs the authentication and servlet filters can perform additional authentication, auditing, or logging tasks.

To perform pre-login and post-login actions using servlet filters, configure these servlet filters for either form login page or for /j_security_check URL. The j_security_check is posted by the form login page with the j_username parameter that contains the user name and the j_password parameter that contains the password. A servlet filter can use user name and password information to perform more authentication or meet other special needs.

1. Develop a form login page and error page for the application.
   Refer to the *Securing applications and their environment* PDF for details.
2. Configure the form login page and the error page for the application as described in .
   Refer to the *Securing applications and their environment* PDF for details.
3. Develop servlet filters if additional processing is required before and after form login authentication.
   Refer to the *Securing applications and their environment* PDF for details.
4. Configure the servlet filters that are developed in the previous step for either the form login page URL or for the /j_security_check URL. Use an assembly tool or development tools like Rational® Application Developer to configure filters. After configuring the servlet filters, the `web-xml` file contains two stanzas.

The first stanza contains the servlet filter configuration, the servlet filter, and its implementation class. The second stanza contains the filter mapping section and a mapping of the servlet filter to the URL.

For more information, see the *Securing applications and their environment* PDF.

Results

This migration results in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

What to do next

The new application uses form-based login and servlet filters to replace the CustomLoginServlet class. Servlet filters also are used to perform additional authentication, auditing, and logging.

Migrating Java 2 security policy

Use this topic for guidance pertaining to migrating Java 2 security policy.

About this task

Previous WebSphere Application Server releases

WebSphere Application Server uses the Java 2 security manager in the server runtime to prevent enterprise applications from calling the System.exit and the System.setSecurityManager methods. These two Java application programming interfaces (API) have undesirable consequences if called by enterprise applications. The System.exit API, for example, causes the Java virtual machine (application server process) to exit prematurely, which is not a beneficial operation for an application server.

To support Java 2 security properly, all the server runtime must be marked as `privileged` (with doPrivileged API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions that are defined in the policy files. The doPrivileged instrumentation is important and necessary to support Java 2 security. Without it, the application code must be granted the permissions that are required by the server runtime. This situation is due to the design and algorithm that is used by Java 2 security to enforce permission checks. Refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the Java 2 security manager (hard coded) for WebSphere Application Server:
- java.lang.RuntimePermission(exitVM)
- java.lang.RuntimePermission(setSecurityManager)

Application code is denied access to these permissions regardless of what is in the Java 2 security policy. However, the server runtime is granted these permissions. All the other permission checks are not enforced.

Only two permissions are supported:
- java.net.SocketPermission
- java.net.NetPermission

However, not all the product server runtime is properly marked as privileged. You must grant the application code all the other permissions besides the two listed previously or the enterprise application can potentially fail to run. This Java 2 security policy for enterprise applications is liberal.

What changed

Java 2 Security is fully supported in WebSphere Application Server, which means that all permissions are enforced. The default Java 2 security policy for an enterprise application is the recommended permission set defined by the Java Platform, Enterprise Edition (Java EE) Version 1.4 specification. Refer to the *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy file for the default Java 2 security policy that is granted to enterprise applications. This policy is a much more stringent compared to previous releases.

All policy is declarative. The product security manager honors all policy that is declared in the policy files. There is an exception to this rule: enterprise applications are denied access to permissions that are declared in the *profile_root*/config/cells/*cell_name*/filter.policy file.

Note: The default Java 2 security policy for enterprise applications is much more stringent and all the permissions are enforced in WebSphere Application Server Version 6.0.x and later. The security policy might fail because the application code does not have the necessary permissions granted where system resources, such as file I/O, can be programmatically accessed and are now subject to the permission checking.

In application code, do not use the setSecurityManager permission to set a security manager. When an application uses the setSecurityManager permission, there is a conflict with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for RMI purposes, you also must enable the **Use Java 2 security to restrict application access to local resources** option on the Global security page within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager. The application code can verify that this security manager is registered by using System.getSecurityManager() application programming interface (API).

Migrating system properties

The following system properties are used in previous releases in relation to Java 2 security:

- **java.security.policy**. The absolute path of the policy file (action required). This system property contains both system permissions (permissions granted to the Java virtual machine (JVM) and the product server runtime) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to WebSphere Application Server Version 6.0.x. For Java 2 security policy migration, see the steps for migrating Java 2 security policy.
- **enableJava2Security**. Used to enable Java 2 security enforcement (no action required). This system property is deprecated; a flag in the WebSphere configuration application programming interface (API) is used to control whether to enable Java 2 security. Enable this option through the administrative console.
- **was.home**. Expanded to the installation directory of WebSphere Application Server (action might be required). This system property is deprecated; superseded by the ${user.install.root} and ${was.install.root} properties. If the directory contains instance-specific data then ${user.install.root} is used; otherwise ${was.install.root} is used. Use these properties interchangeably for the WebSphere Application Server or the Network Deployment environments. See the steps for migrating Java 2 security policy.

Migrating the Java 2 Security Policy

No easy way exists to migrate the Java policy file to WebSphere Application Server Version 6.0.*x* and later automatically because of a mixture of system permissions and application permissions in the same policy file. Manually copy the Java 2 security policy for enterprise applications to a was.policy or app.policy file. However, migrating the Java 2 security policy to a was.policy file is preferable because symbols or relative code base is used instead of an absolute code base. This process has many advantages. Grant the permissions that are defined in the was.policy to the specific enterprise application only, while permissions in the app.policy file apply to all the enterprise applications that run on the node where the app.policy file belongs.

Refer to the *Securing applications and their environment* PDF for more details on policy management.

The following example illustrates the migration of a Java 2 security policy from a previous release. The contents include the Java 2 security policy file for the app1.ear enterprise application and the system permissions, which are permissions that are granted to the Java virtual machine (JVM) and the product server runtime.

The default location for the Java 2 security policy file is *profile_root*/properties/java.policy. Default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${app_server_root}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${app_server_root}${/}temp${/}somefile.txt",
       "read";
   };
```

For clarity of illustration, all the permissions are migrated as the application level permissions in this example. However, you can grant permissions at a more granular level at the component level (Web, enterprise beans, connector or utility Java archive (JAR) component level) or you can grant permissions to a particular component.

1. Ensure that Java 2 security is disabled on the application server.
2. Create a new was.policy file, if the file is not present, or update the was.policy file for migrated applications in the configuration repository with the following contents:

   ```
   grant codeBase "file:${application}" {
        permission java.security.SecurityPermission "printIdentity";
        permission java.io.FilePermission "
                ${user.install.root}${/}temp${/}somefile.txt", "read";
      };
   ```

   The third and fourth lines in the previous code sample are presented on two lines for illustrative purposes only.

   The was.policy file is located in the *profile_root*/config/cells/*cell_name*/applications/app.ear/deployments/app/META-INF/ directory.
3. Use an assembly tool to attach the was.policy file to the enterprise archive (EAR) file.

   You also can use an assembly tool to validate the contents of the was.policy file. For more information, see the *Securing applications and their environment* PDF.
4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 security permissions and the default permissions set declared in the ${user.install.root}/config/cells/*cell_name*/nodes/*node_name*/app.policy file. This validation requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction environment. Refer to developer kit APIs protected by Java 2 security for information about which APIs are protected by Java 2 security. If you use third-party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.
5. Perform preproduction testing of the migrated enterprise application with Java 2 security enabled. Enable trace for the WebSphere Application Server Java 2 security manager in a preproduction testing environment with the following trace string: com.ibm.ws.security.core.SecurityManager=all=enabled. This trace function can be helpful in debugging the AccessControlException exception that is created when an application is not granted the required permission or some system code is not properly marked as privileged. The trace dumps the stack trace and permissions that are granted to the classes on the call stack when the exception is created.

   For more information, see the *Securing applications and their environment* PDF.

   Note: Because the Java 2 security policy is much more stringent compared with previous releases, the administrator or deployer must review their enterprise applications to see if extra

permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

Migrating with Tivoli Access Manager for authentication enabled

When Tivoli® Access Manager security is configured for your existing environment and security is enabled, you can migrate to WebSphere Application Server, Version 7.0.

Before you begin

Your profiles must be migrated using the migration tools to migrate product configurations.

Note: Do not restart the WebSphere Application Server Version 7.0 server until after performing the following procedure. The migration tools omit some files that enable the server to start correctly.

About this task

After migrating your profiles additional steps are required when Tivoli Access Manager security is configured.

1. Copy the profile_root1/PolicyDirector directory and it's contents to profile_root2/PolicyDirector. For this example:
   - *profile_root1* is the root directory of the profile being migrated.
   - *profile_root2* is the root directory of the version 6.1 profile.
   a. From an i5/OS command line, type **STRQSH** and press **Enter**.
   b. Type `cp -R profile_root1/PolicyDirector profile_root2` and press **Enter**.
2. Copy the key file of the profile being migrated to the version 7.0 profile. The location of the key file is defined in profile_root1/PolicyDirector/PdPerm.properties. For this example:
   - The `PdPerm.properties` file contains *pdcert-url=file\:/QIBM/UserData/WebAS51/Base/AppSvr1/etc/AppSvr1.kdb*.
   - /QIBM/UserData/WebAS51/Base/AppSvr1 is the root directory of a Version 5.1 profile.
   a. From an i5/OS command line type **STRQSH** and press **Enter**.
   b. Type `cp /QIBM/UserData/WebAS51/Base/AppSvr1/etc/AppSvr1.kdb profile_root2/etc/AppSvr1.kdb` and press **Enter**.
3. Edit the property values in profile_root2/PolicyDirector/PdPerm.properties and in profile_root2/PolicyDirector/Pd.properties to replace occurrences of *profile_root1* with *profile_root2* in the file path name values.

Migrating Java thin clients that use the password encoding algorithm

To migrate Java thin clients that are enabled for OS400 password encoding, use the following information to modify the Java client invocation so that the os400.security.password properties are no longer set on the invocation.

About this task

The password encoding feature offers the following encoding algorithms:

- XOR, which is the default
- OS400

In Version 5 and later, the value of the os400.security.password.validation.list.object property is dependant upon the property value passed to the thin client using the JAVA_FLAGS environment variable. The JAVA_FLAGS environment variable is set by the **setupClient** script. The **setupClient** script calls the **setupCmdLine** script, which is where the value for the os400.security.password.validation.list.object

property is set. For example, if a Version 6.*x* Base Edition Java client is passed **-profileName default**, then the **setupClient** script calls the *profile_root*/default/bin/setupCmdLine file.

To migrate Java thin clients that are enabled for OS400 password encoding, modify the Java client invocation so that the os400.security.password properties are no longer set on the invocation. The following code sample does not contain the os400.security.password properties:

```
java -classpath $MY_CLIENT_CLASSES:app_server_root/classes/wsa400.jar:$WAS_CLASSPATH \
  $CLIENTSAS $JAVA_FLAGS \
  -Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory \
  -Djava.naming.provider.url=iiop://server1:10151 \
  MyClientClass $*
```

Perform the following steps if the following condition is true:

- If the passwords in the sas.client.props file for that profile are encoded with the OS400 password encoding algorithm

1. Replace all of the OS400 encoded passwords, which have {OS400}) prefixes in the sas.client.props file for the Application Server profile, with the clear text values of the passwords.
2. Encode the passwords using the **PropFilePasswordEncoder** Qshell command.

   For more information, see PropFilePasswordEncoder command reference.

Results

Note: You can configure a WebSphere Application Server profile to encode passwords with the XOR algorithm even though the profile is enabled to decode passwords that were encoded with either the OS400 algorithm or the XOR algorithm. If you encode these passwords with the XOR algorithm, then the passwords in the sas.client.props file are encoded with the XOR algorithm.

Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default, and you need to configure it for use.

About this task

Enabling Tivoli Access Manager security within WebSphere Application Server requires:

- A supported Lightweight Directory Access Protocol (LDAP) installed somewhere on your network. This user registry contains the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- Tivoli Access Manager server exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager, refer to the IBM Tivoli Access Manager for e-business information center.

  Note: WebSphere Application Server contains an embedded client for Tivoli Access Manager. To use Tivoli Access Manager, you must also configure the Tivoli Access Manager server.

  However, the server version must be the same version or later as the client version. For information on the supported version of Tivoli Access Manager, see WebSphere Application Server - Supported Prerequisites.
- WebSphere Application Server is installed either in a single server model or as WebSphere Application Server Network Deployment.
- When administrative security is configured with a Federal Information Processing Standard (FIPS) provider, the Tivoli Access Manager server must be configured for FIPS as well

Complete the following steps to enable embedded Tivoli Access Manager security:

1. Create the security administrative user.

For more information, see the *Securing applications and their environment* PDF.

2. Configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager .

   For more information, see the *Securing applications and their environment* PDF.

3. Enable WebSphere Application Server security. When you are using Tivoli Access Manager you must configure LDAP as the user registry.

   For more information, see the *Securing applications and their environment* PDF.

4. Enable the JACC provider for Tivoli Access Manager.

   For more information, see the *Securing applications and their environment* PDF.

Propagating security policy of installed applications to a JACC provider using wsadmin scripting

It is possible that you have applications installed prior to enabling the Java Authorization Contract for Containers (JACC)-based authorization. You can start with default authorization and then move to an external provider-based authorization using JACC later.

Before you begin

Note: Use the wsadmin tool to propagate information to the JACC provider independent of the application installation process, avoiding the need to reinstall applications. Also, during application installation or modification you might have had problems propagating the security policy information to the JACC provider. For example, network problems might occur, the JACC provider might not be available, and so on. For these cases, the security policy of the previously installed applications does not exist in the JACC provider to make the access decisions. One choice is to reinstall the applications involved. However, you can avoid reinstalling by using the wsadmin scripting tool. Use this tool to propagate information to the JACC provider independent of the application installation process. The tool eliminates the need for reinstalling the applications.

The tool uses the SecurityAdmin MBean to propagate the policy information in the deployment descriptor of any installed application to the JACC provider. You can invoke this tool using wsadmin at the base application server for base and deployment manager level for Network Deployment. Note that the SecurityAdmin MBean is available only when the server is running.

Use `propagatePolicyToJACCProvider{-appNames appNames}` to propagate the policy information in the deployment descriptor or annotations of the enterprise archive (EAR) files to the JACC provider. If the RoleConfigurationFactory and the RoleConfiguration interfaces are implemented by the JACC provider, the authorization table information in the binding file of the EAR files is also propagated to the provider. See the *Securing applications and their environment* PDF for more information about these interfaces.

The `appNames String` contains the list of application names, delimited by a colon (:), whose policy information must be stored in the provider. If appNames is not present, the policy information of all the deployed applications is propagated to the provider.

Also, be aware of the following items:

- Before migrating applications to the Tivoli Access Manager JACC provider, create or import the users and groups that are in the applications to Tivoli Access Manager.
- Depending on the application or the number of applications that are propagated, you might have to increase the request time-out period either in the `soap.client.props` file in the directory *profile_root*`/properties` (if using SOAP) or in the `sas.client.props` file (if using RMI) for the command to complete. You can set the request time-out value to 0 to avoid the timeout problem, and change it back to the original value after the command is run.

1. Configure your JACC provider in WebSphere Application Server.

   See the *Securing applications and their environment* PDF for more information.

2. Restart the server.
3. Enter the following commands:

```
wsadmin>$AdminTask propagatePolicyToJACCProvider {-appNames appNames}
```

JACCUtilityCommands command group for the AdminTask object

Use this topic as a reference for the commands for the JACCUtilityCommands group for the AdminTask object. Use these commands to determine whether Java Authorization Contract for Containers (JACC) is enabled and whether the runtime uses a single security domain. You can also use these commands to propagate the security policies for application to the JACC provider.

The following commands are available for the JACCUtilityCommands group of the AdminTask object.

- "isJACCEnabled"
- "isSingleSecurityDomain"
- "propagatePolicyToJACCProvider " on page 81

isJACCEnabled

The isJACCEnabled command displays whether JACC is enabled or disabled in the global security domain when the server was started. The command does not indicate dynamic changes. Instead, it displays the JACC status at server startup.

Target object

None.

Required parameters

None.

Return value

The command returns `true` if JACC is enabled. The command returns `false` if JACC is disabled.

Batch mode example usage

Using Jython string:

```
AdminTask.isJACCEnabled()
```

Interactive mode example usage

Using Jython:

```
AdminTask.isJACCEnabled('-interactive')
```

isSingleSecurityDomain

The isSingleSecurityDomain command displays whether the environment is configured to use a single security domain when the server was started. The command does not indicate dynamic changes. Instead, it displays the security domain status at server startup.

Target object

None.

Required parameters

None.

Return value

The command returns `true` if the environment uses a single security domain. The command returns the `false` string if the environment uses multiple security domains.

Batch mode example usage

Using Jython:

```
AdminTask.isSingleSecurityDomain()
```

Interactive mode example usage

Using Jython:

```
AdminTask.isSingleSecurityDomain('-interactive')
```

propagatePolicyToJACCProvider

The propagatePolicyToJACCProvider command propagates the security policies of the applications of interest to the JACC provider. This command is supported in a single security domain environment only.

Target object

None.

Required parameters

None.

Optional parameters

-appNames
Specifies a list of application names delimited with a colon character (`:`). (String, optional)

The command uses all applications if you do not specify a value for this parameter, as the following syntax demonstrates: `AdminTask.propagatePolicyToJACCProvider()`

Return value

The command does not return output.

Batch mode example usage

Using Jython string:

```
AdminTask.propagatePolicyToJACCProvider ('–appNames "app1:app2:app3"')
```

Using Jython list:

```
AdminTask.propagatePolicyToJACCProvider ('–appNames', '"app1:app2:app3"')
```

Interactive mode example usage

Using Jython:

```
AdminTask.propagatePolicyToJACCProvider ('-interactive')
```

Chapter 9. Naming and directory

Migrating to Java Platform, Standard Edition (Java SE) 6

This product version supports the Java Platform, Standard Edition (Java SE) 6 specification. Its Java virtual machine provides a Java language compiler and runtime environment. Decide whether your new and existing applications will take advantage of the capabilities added by Java SE 6, adjust the just-in-time (JIT) mode if necessary, and begin the transition from deprecated functions.

About this task

The following JSRs are new in Java SE 6:

- JSR 105: XML Digital Signature Application Programming Interfaces (APIs)
- JSR 173: Streaming API for XML (StAX)
- JSR 181: Web Services Metadata
- JSR 199: Java Compiler API
- JSR 202: Java Class-File Specification Update
- JSR 221: Java DataBase Connectivity (JDBC) 4.0
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0
- JSR 223: Scripting for the Java Platform
- JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0
- JSR 250: Common Annotations
- JSR 269: Pluggable Annotation-Processing API

The new virtual machine specification adds several features and functions to benefit application developers, such as interfaces for integrating the Java and scripting languages, password prompting, file input-output enhancements, and parsing of streaming XML documents.

- Decide whether to take advantage of new Java SE 6 capabilities in your applications.

  You can deploy applications using Java SE 6 features only to Version 7 nodes, as earlier product versions do not provide the Java SE 6 virtual machine.

  Applications that access classes and APIs internal to the Java virtual machine might produce errors. These classes and APIs are not covered by the Java SE 6 specification and are therefore subject to change. Packages with prefixes such as com.sun.* are considered internal. Additionally, direct use of implementations of XML and XSL parsers is strongly discouraged, such as direct use of Xerces and Xalan classes that provide the Java API for XML Processing (JAXP) implementation for the virtual machine. The direct parser APIs also are considered internal and subject to change. Applications should rely only on the JAXP APIs defined in the Java SE 6 API documentation. If your application requires a specific version of Xerces or Xalan, or some other XML/XSL parser package, then embed the parser within your application's `WEB-INF/lib` directory and set the appropriate class loading mode in your application deployment so that for your application the XML parser APIs are loaded from the application class path, not the Java virtual machine bootstrap class path. Failure to follow this guideline can cause significant errors when you try to migrate to a new Java SE 6 level.
- Compile Java SE 6 applications to run on previous Java virtual machine levels by setting the compiler modes.

  When compiling applications that are built with Java SE 6 that are intended for running on previous specifications, specify `-source` and `-target` modes for the Java SE 6 compiler. Doing so ensures that the bytecode generated is compatible with the earlier Java virtual machine.

  For example, if the target Java virtual machine is at 1.4.2 level, when you compile applications with Java SE 6, you should specify `-source 1.4`, and `target 1.4` to generate bytecode compatible with 1.4.2. This does not handle the usage of packages, classes, or functions new to Java SE 6. It only

addresses bytecode output. Developers must take care in what APIs they are using from the J2SE packages if they intend to run the application on multiple Java virtual machine specification levels.

- Address incompatibilities in previously compiled Java 2 Standard Edition (J2SE) 1.4 and 5.0 applications.

  Java SE 6 is upwards binary-compatible with Java 2 Technology Edition, Version 5.0 and Java 2 Technology Edition, Version 1.4.2, except for the incompatibilities documented by Sun Microsystems at http://java.sun.com/javase/technologies/compatibility.jsp.

- Transition from deprecated Java Virtual Machine Debug Interface (JVMDI) and Java Virtual Machine Profiler Interface (JVMPI) functions to Java Virtual Machine Tool Interface (JVMTI).

  JVMDI and JVMPI functions were deprecated in J2SE 5.0. They have been removed from Java SE 6.

- Update your use of the Java command line interface.

  The command-line interfaces for the Java SE 6 level have not changed extensively from J2SE 5, although they vary among virtual machine vendors. You can find them in the `JAVA_HOME/bin` directory. Here are some notable command line options that are standard to all Java SE 6 implementations.

  – For JVMTI, use `-agentlib` to load a native agent library that you specify.
  – For JVMTI, use `-agentpath` to load the native agent library by the full path name.
  – For JVMTI, use `-javaagent` to load the Java programming language agent (see java.lang.instrument for details).
  – See `apt -help` for information about this new command line supporting the annotations capability.
  – See `javac -help` for information and updates to that command line.

- Update ANT tasks.

  If you have created ANT tasks based on the idltojava ANT task shipped with prior versions of this product, ensure that it passes the proper parameters for Java SE 6 as it does for J2SE 1.4 or 5, to ensure the stubs, ties and skeletons that it generates are compatible with earlier product releases.

JNDI interoperability considerations

You must take extra steps to enable your programs to interoperate with non-product JNDI clients and to bind resources from MQSeries® to a namespace.

EJB clients running in an environment other than WebSphere Application Server accessing EJB applications running on WebSphere Application Server servers

When an enterprise bean (EJB) application running in WebSphere Application Server is accessed by a non-product EJB client, the JNDI initial context factory is presumed to be a non-product implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, the corbaname format can be used to look up the EJB home.

Single server

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in the server **server1** in node **node1** and bound in that server under the name **myEJB**:

```
initialContext.lookup(
  "corbaname:iiop:myHost:2809#cell/nodes/node1/servers/server1/myEJB");
```

Without CORBA object URL support

If the JNDI initial context factory being used does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as follows:

```
Hashtable env = new Hashtable();
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");
Context ic = new InitialContext(env);
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

Binding resources from MQSeries 5.2

In releases previous to WebSphere Application Server Version 5.0, the MQSeries jmsadmin tool could be used to bind resources to the namespace. When used with a WebSphere Application Server namespace, the resource is bound within a transient partition in the namespace and does not persist past the life of the server process. Instead of binding the MQSeries resources with the jmsadmin tool, bind them from the administrative console, under **Resources** in the console navigation tree.

Chapter 10. Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

Your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application.

Application profiling

Running Version 5 Application Profiles on Version 7.0

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, you must change a setting. Also, there are several implications to using version 5.x application profiles on version 7.0.

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 *Developing and deploying applications* PDF for more information.

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

  See the *Developing and deploying applications* PDF.

  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 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 *Developing and deploying applications* PDF.
- 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 *Developing and deploying applications* PDF 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.

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.

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.

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 7 from previous product releases, read the Migrating product configurations topic.

1. Install the Version 7 product. Installing the product creates a stand-alone application server.
2. Migrate the previous release to the Version 7 product.

   Read the Overview of migration, coexistence, and interoperability topic.

Appendix. Directory conventions

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

Default product locations - IBM i

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

app_client_root
The default installation root directory for the Java EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V7/client directory.

app_client_user_data_root
The default Java EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V7/client directory.

app_client_profile_root
The default Java EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V7/client/profiles/*profile_name* directory.

app_server_root
The default installation root directory for WebSphere Application Server - Express is the /QIBM/ProdData/WebSphere/AppServer/V7/Express directory.

cip_app_server_root
The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_profile_root
The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_user_data_root
The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

if_root This directory represents the root directory of the IBM WebSphere Installation Factory. Because you can download and unpack the Installation Factory to any directory on the file system to which you have write access, this directory's location varies by user. The Installation Factory is an Eclipse-based tool which creates installation packages for installing WebSphere Application Server in a reliable and repeatable way, tailored to your specific needs.

iip_root
This directory represents the root directory of an *integrated installation package* (IIP) produced by the IBM WebSphere Installation Factory. Because you can create and save an IIP to any directory on the file system to which you have write access, this directory's location varies by user. An IIP is an aggregated installation package created with the Installation Factory that can include one or more generally available installation packages, one or more customized installation packages (CIPs), and other user-specified files and directories.

java_home
The following directories are the root directories for all supported Java Virtual Machines (JVMs).

JVM	Directory
Classic JVM	/QIBM/ProdData/Java400/jdk6
32–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit
64–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit

plugins_profile_root
The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver/profiles/*profile_name* directory.

plugins_root
The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V7/webserver directory.

plugins_user_data_root
The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver directory.

product_library
product_lib
This is the product library for the installed product. The product library for each Version 7.0 installation on the system contains the program and service program objects (similar to .exe, .dll, .so objects) for the installed product. The product library name is QWAS7*x* (where *x* is A, B, C, and so on). The product library for the first WebSphere Application Server Version 7.0 product installed on the system is `QWAS7A`. The *app_server_root*/properties/product.properties file contains the value for the product library of the installation, `was.install.library`, and is located under the *app_server_root* directory.

profile_root
The default directory for a profile named *profile_name* for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express/profiles/*profile_name* directory.

shared_product_library
The shared product library, which contains all of the objects shared by all installations on the system, is `QWAS7`. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

updi_root
The default installation root directory for the Update Installer for WebSphere Software is the /QIBM/ProdData/WebSphere/UpdateInstaller/V7/updi directory.

user_data_root
The default user data directory for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express directory.

The profiles and profileRegistry subdirectories are created under this directory when you install the product.

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

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.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.