

IBM WebSphere Application Server Network Deployment
for IBM i, Version 8.5

Developing and deploying applications

IBM

Note

Before using this information, be sure to read the general information under “Notices” on page 259.

Compilation date: June 7, 2012

© Copyright IBM Corporation 2012.

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

Contents

How to send your comments	vii
Using this PDF	ix
Chapter 1. Overview and new features for developing applications	1
Chapter 2. How do I develop applications?	3
Migrating to Java Platform, Enterprise Edition (Java EE) 6	3
Migrating to Java Platform, Standard Edition (Java SE) 6	4
Chapter 3. Designing applications	7
Chapter 4. Obtaining an integrated development environment (IDE)	9
Chapter 5. Debugging applications	11
Debugging components in the IBM Rational Application Developer for WebSphere	12
Debugging Service details	13
Enable service at server startup	13
JVM debug port	13
JVM debug arguments	13
Debug class filters	13
Chapter 6. Assembling applications	15
Application assembly and enterprise applications	16
Development and assembly tools	17
Generating code for web service deployment	17
Assembling applications: Resources for learning	18
Chapter 7. Class loading	21
Class loaders	21
Configuring class loaders of a server	25
Class loader collection	27
Class loader ID	27
Class loader order	27
Class loader settings	27
Configuring application class loaders	28
Configuring web module class loaders	30
Class loading: Resources for learning	31
Chapter 8. Adding logging and tracing to your application	33
Using Java logging in an application	34
Using a logger	35
Java logging	44
Configuring the logger hierarchy	45
Creating log resource bundles and message files	46
Logger.properties file for configuring logger settings	48
Configuring applications to use Jakarta Commons Logging	49
Jakarta Commons Logging	50
Configurations for the WebSphere Application Server logger	53
Programming with the JRas framework	55
JRas logging toolkit	55
JRas Extensions	57
JRas messages and trace event types	65

Instrumenting an application with JRas extensions	68
Logging Common Base Events in WebSphere Application Server	74
The Common Base Event in WebSphere Application Server	75
Logging with Common Base Event API and the Java logging API	88
java.util.logging -- Java logging programming interface	98
Logger.properties file	99
Logging Common Base Events in WebSphere Application Server	100
Showlog commands for Common Base Events	100
Chapter 9. Overview and new features for deploying applications	101
Chapter 10. Deploying applications to the Liberty profile	103
Packaging a Liberty profile server from the command prompt	104
Using JNDI binding for constants from the server configuration files	105
Sharing common OSGi bundles for the Liberty profile	106
Configuring class loaders for Java EE applications	106
Using a Java library with a Java EE application	107
Sharing a library across multiple Java EE applications	107
Accessing third-party APIs from a Java EE application	108
Removing access to third-party APIs for a Java EE application	109
Overriding a provided API with an alternative version	109
Providing global libraries for all Java EE applications	110
Deploying data access applications to the Liberty profile	111
Deploying an existing JDBC application to the Liberty profile	111
Enabling JDBC Tracing for the Liberty profile	112
Deploying a web application to the Liberty profile	115
Deploying a JPA application to the Liberty profile	116
Chapter 11. How do I deploy applications?.	119
Chapter 12. Deploying enterprise applications	121
Installing enterprise application files	121
Installable enterprise module versions	122
Ways to install enterprise applications or modules	124
Installing enterprise application files with the console	126
Example: Installing an EAR file using the default bindings	133
Example: Installing a web services sample with the console	134
Preparing for application installation settings	135
Preparing for application installation binding settings	136
Select installation options settings	142
Manage modules settings	152
Client module settings	154
Client module property settings	155
Provide options to compile JavaServer Pages settings	155
EJB JNDI names for beans	156
Bind EJB business settings	157
Map default data sources for modules containing 1.x entity beans	158
EJB references	159
Resource references	160
Virtual hosts settings	163
Security role to user or group mapping	164
JASPI authentication enablement for applications	165
User RunAs collection	166
Ensure all unprotected 1.x methods have the correct level of protection	166
Bind listeners for message-driven beans settings	167
Map data sources for all 2.x CMP beans	169

Map data sources for all 2.x CMP beans settings	171
Ensure all unprotected 2.x methods have the correct level of protection	173
Provide options to perform the EJB Deploy settings	174
Shared library reference and mapping settings	176
Shared library relationship and mapping settings	177
JSP and JSF option settings	178
Context root for web modules settings	180
Initial parameters for servlets settings	180
Environment entries for client modules settings	181
Environment entries for EJB modules settings	182
Environment entries for web modules settings	183
Environment entries for application settings	183
Resource environment references	184
Message destination reference settings	185
Select current backend ID settings.	185
Provide JNDI names for JCA objects settings.	186
Correct use of the system identity	186
Requirements for setting data access isolation levels	187
Metadata for module settings.	189
Provide options to perform the web services deployment settings	191
Display module build ID settings	192
Task overview: Assembling applications using remote request dispatcher	192
Installing enterprise modules with JSR-88	194
Customizing modules using DConfigBeans	196
Chapter 13. Deploying and administering business-level applications	199
Business-level applications	199
Assets	202
Composition units	202
Importing assets	203
Upload asset settings	205
Asset settings	206
Managing assets	209
Asset collection.	210
Updating assets	211
Deleting assets	215
Exporting assets	215
Creating business-level applications	216
Creating business-level applications with the console	217
Business-level application settings	227
Composition unit settings	229
Example: Creating a business-level application	232
Starting business-level applications	233
Stopping business-level applications	234
Updating business-level applications	235
Deleting business-level applications	237
Chapter 14. Troubleshooting deployment	239
Application deployment problems	239
Application deployment troubleshooting tips	245
Application startup errors	245
Application startup problems	251
Reducing annotation searches during application deployment.	253
A client program does not work	254
Web resource is not displayed	255
Application uninstallation problems.	257

Notices	259
Trademarks and service marks	261
Index	263

How to send your comments

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

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

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

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

Using this PDF

Links

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

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

Print sections directly from the information center navigation

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

To print a section of the information center navigation:

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

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

Chapter 1. Overview and new features for developing applications

View the topics in the following list to learn more about developing applications for deployment on this product.

What is new for developers

This topic provides an overview of new and changed features of the programming model and application serving environment as it pertains to development and test efforts.

Learn about WebSphere applications: Overview and new features






This topic provides an overview of the programming model.

Accessing the samples

The samples are a good way to become familiar with the programming model.

Chapter 2. How do I develop applications?

Follow these shortcuts to get started quickly with popular tasks.

-  Design applications
-  Automate the build environment with Apache Ant
-  Secure applications, messages, and data
-  Learn about WebSphere programming extensions
-  Add logging and tracing to applications.

Migrating to Java Platform, Enterprise Edition (Java EE) 6

Version 8.5 of the product supports the Java Platform, Enterprise Edition (Java EE) 6 specification. Your new and existing enterprise applications can take advantage of the capabilities added by Java EE 6.

About this task

The product supports the following specification and application programming interface (API) levels that are new in Java EE 6:

- JSR 318: Enterprise JavaBeans (EJB) 3.1
- JSR 315: Java Servlet 3.0
- JSR 245 JavaServer Pages/Expression Language (JSP/EL) 2.1
- JSR 314: JavaServer Faces (JSF) 2.0
- JSR 199: JMS 1.1
- JSR-299: Java Contexts and Dependency Injection (JCDI) 1.0 (was Web Beans)
- JSR 317: Java Persistence API (JPA) 2.0
- JSR 322: Java EE Connector Architecture (JCA) 1.6
- Java API for XML-Based Web Services (JAX-WS) 2.2
- JSR 311: Java API for RESTful Web Services (JAX-RS) 1.0
- JSR 196: Java Authentication Service Provider Interface for Containers (JASPIC) 1.0
- JSR 303: Bean Validation 1.0

The new specifications add several capabilities to benefit application developers, such as profiles that provide common features among applications.

Further, several specifications expand the use of annotations to more module types. Java language annotations simplify development of Java EE applications. By using annotations, many applications can avoid the need for deployment descriptors. In Version 7, the product supported annotations for EJB 3.0 and Web 2.5 modules. In Version 8, the product supports annotations for additional module types, such as resource adapters or RAR files, as well as continues to support the use of deployment descriptors.

The general steps for migrating your enterprise applications follow.

Procedure

1. Decide whether to take advantage of new Java EE 6 capabilities in your applications.

The Version 8.5 product supports applications written to Java EE 6 and supports portable applications written to previous Java EE versions, specifically Java EE 5, Java 2 Platform, Enterprise Edition (J2EE) 1.4, and J2EE 1.3. If you decide not to use new Java EE 6 capabilities, your portable applications will continue to work without change and with identical behavior on the current version of the platform.

2. If you select to use new Java EE 6 capabilities in your applications, change the applications as needed to conform to the specifications.
3. Deploy your applications.

Deploy applications that use new Java EE 6 capabilities only to Version 8 deployment targets. You can deploy applications written to previous specifications to Version 8 deployment targets or to Version 6.x or 7.x deployment targets.

What to do next

Test the deployed applications to ensure that the applications behave as expected. Update the applications as needed.

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.

The Java Monitoring and Management Console (JConsole) is part of the Java Development Kit (JDK) and the IBM Software Development Kit (SDK) Version 6. Although these development kits are shipped with WebSphere Application Server, the product does not support the JConsole tool.

Procedure

- 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 or later 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. 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 Oracle Corporation at <http://java.sun.com/javase/technologies/compatibility.jsp>.

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

Note: JVMDI and JVMPPI 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 `idtojava` 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.

Chapter 3. Designing applications

This topic highlights websites and other ideas for finding best practices for designing WebSphere® applications, particularly in the realm of WebSphere extensions to the Java Platform, Enterprise Edition (Java EE) specification.

When designing WebSphere applications, follow the example set by the Samples. Refer to the code in the Samples Gallery that is available with the product. In particular, the Samples Gallery highlights new and WebSphere-specific aspects of the programming model.

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

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

Web resources for learning

- The top 10 (more or less) J2EE best practices

The authors, who are IBM consultants and performance experts, describe this document in the following way: Over the last five years, a lot has been written about Java EE best practices. There now are probably 10 or more books along with dozens of articles that provide insight into how Java EE applications should be written. In fact, there are so many resources, often with contradictory recommendations, navigating the maze has become an obstacle to adopting Java EE itself. To provide some simple guidance for customers entering this maze, we set out to compile the following "top 10" list of what we feel are the most important best practices for J2EE.

- IBM Patterns for e-Business

Patterns for e-business are a group of reusable assets that can help speed the process of developing Web-based applications. The patterns leverage the experience of IBM architects to create solutions quickly, whether for a small local business or a large multinational enterprise.

- Best practices for using XSLT in WebSphere Application Server applications

The author states: In this article I explore the reasons why some WebSphere Application Server applications use XSL for HTML production instead of JavaServer Pages (JSP) files. I will compare the performance of XSLT for HTML/XHTML production against JSP files and browser formatting. I will then provide guidance on how to improve XSLT performance in WebSphere Application Server should you decide to go this route. While this article focuses on the use of XSLT for the production of HTML, the performance best practices are directly applicable to other WebSphere Application Server uses of XSLT, such as XML-to-XML transformations and XML-to-text transformations.

- Rational on developerWorks

The developerWorks® site provides quick links to technical resources and best practices for Rational® software. Browse information by product or by technology. Find resources for learning, support, and developer communities.

- developerWorks site

developerWorks is an IBM technical resource for developers, providing a wide range of tools, code, and education on DB2®, eServer™, Lotus®, Rational, Tivoli®, and WebSphere as well as on open standards technology such as web services, Wireless, Linux, XML, Java technologies, and more. By providing focused and relevant technical information for developers, developerWorks offers choices you can apply to building and deploying applications across heterogeneous systems. Using developerWorks, you can take full advantage of open standards and the IBM Software Development Platform in an on demand world.

- Resource reference list

The product has a large amount of existing documentation. Use the following user communities and other non-IBM sites that gather knowledge about using WebSphere products as a guideline to find the documentation that you require.

- <http://www.websphere-world.com/>
- <http://www.websphere.org/>
- <http://www.webspherepro.com/wphome/>
- <http://www.sys-con.com/websphere/>
- <http://websphereadvisor.com/>

See also the documentation for the type of application that you are developing, such as web applications, EJB applications, Web services applications, or applications that use messaging.

Chapter 4. Obtaining an integrated development environment (IDE)

This topic describes obtaining an integrated development environment (IDE). Use Rational products from IBM to design, construct, and manage changes to applications for deployment on your WebSphere Application Server products.

Procedure

- See “Development and assembly tools” on page 17 for a description of the Rational Application Developer product.

Insert the product disc and use the documentation and the installation program on the disc to install and set up the development environment.

- Refer to these web resources for learning.

Rational software pages on ibm.com

Browse the IBM portfolio of software for requirements analysis and tracking, application design and construction, ensuring software quality, configuration and change management, and development project management.

Rational developer community

This page provides quick links to technical resources and best practices for Rational software on developerWorks. Browse information by product or by technology. Find resources for learning, support, and developer communities.

developerWorks main page

This page is the entrance to the IBM resource for developers.

Chapter 5. Debugging applications

To debug your application, you must use a development environment like the IBM Rational Application Developer for WebSphere to create a Java project. You must then import the program that you want to debug into the project.

About this task

By following the steps below, you can import the WebSphere Application Server examples into a Java project. Two debugging styles are available:

- **Step-by-step** debugging mode prompts you whenever the server calls a method on a web object. A dialog lets you step into the method or skip it. In the dialog, you can turn off step-by-step mode when you are finished using it.
- **Breakpoints** debugging mode lets you debug specific parts of programs. Add breakpoints to the part of the code that you must debug and run the program until one of the breakpoints is encountered.


Breakpoints actually work with both styles of debugging. Step-by-step mode just lets you see which web objects are being called without having to set up breakpoints ahead of time.

You do not need to import an entire program into your project. However, if you do not import all of your program into the project, some of the source might not compile. You can still debug the project. Most features of the debugger work, including breakpoints, stepping, and viewing and modifying variables. You must import any source that you want to set breakpoints in.


The inspect and display features in the source view do not work if the source has build errors. These features let you select an expression in the source view and evaluate it.

Procedure

1. Create a Java Project by opening the New Project dialog.
2. Select **Java** from the left side of the dialog and **Java Project** in the right side of the dialog.
3. Click **Next** and specify a name for the project, for example, WASExamples.
4. Click **Finish** to create the project.
5. Select the new project, choose **File > Import > File System**, then **Next** to open the import file system dialog.
6. Browse the directory for files.

 Go to the following directory: `profile_root/installedApps/node_name/DefaultApplication.ear/DefaultWebApplication.war`.

7. Select `DefaultWebApplication.war` in the left side of the Import dialog and then click **Finish**. This imports the JavaServer Pages files and Java source for the examples into your project.
8. Add any JAR files needed to build to the Java Build Path.

 Select **Properties** from the right-click menu. Choose the Java Build Path node and then select the Libraries tab. Click **Add External JARs** to add the following JAR files:

- `profile_root/installedApps/node_name/DefaultApplication.ear/Increment.jar`

When you have added this JAR file, select it and use the **Attach Source** function to attach the `Increment.jar` file because it contains both the source and class files.

- `app_server_root/dev/JavaEE/j2ee.jar`
- `app_server_root/plugins/com.ibm.ws.runtime.jar`
- `app_server_root/plugins/com.ibm.ws.webcontainer.jar`

Click **OK** when you have added all of the JARs.

9. You can set some breakpoints in the source at this time if you like, however, it is not necessary as step-by-step mode will prompt you whenever the server calls a method on a web object. Step-by-step mode is explained in more detail below.

10. To start debugging, you need to start the WebSphere Application Server in debug mode and make note of the JVM debug port. The default value of the JVM debug port is 7777.
11. When the server is started, switch to the debug perspective by selecting **Window > Open Perspective > Debug**. You can also enable the debug launch in the Java Perspective by choosing **Window > Customize Perspective** and selecting the **Debug** and **Launch** checkboxes in the **Other** category.
12. Select the workbench toolbar **Debug** pushbutton and then select **WebSphere Application Server Debug** from the list of launch configurations. Click the **New** pushbutton to create a new configuration.
13. Give your configuration a name and select the project to debug (your new WASEExamples project). Change the port number if you did not start the server on the default port (7777).
14. Click **Debug** to start debugging.
15. Load one of the examples in your browser. For example: `http://your.server.name:9080/hitcount`

IBM i

What to do next

To learn more about debugging, launch The IBM Rational Application Developer for WebSphere, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry. To learn about known limitations and problems that are associated with the IBM Rational Application Developer for WebSphere, see the IBM Rational Application Developer for WebSphere release notes. For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

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

Debugging components in the IBM Rational Application Developer for WebSphere

The IBM Rational Application Developer for WebSphere, included with the WebSphere Application Server on a separately-installable CD, includes debugging functionality that is built on the Eclipse workbench. Documentation for the IBM Rational Application Developer for WebSphere is provided with that product. To learn more about the debug components, launch the IBM Rational Application Developer for WebSphere, select **Help > Help Contents** and choose the **Developing > Debugging applications** bookshelf entries.

The IBM Rational Application Developer for WebSphere includes the following components:

The WebSphere Application Server debug adapter

which allows you to debug web objects that are running on WebSphere Application Server and that you have launched in a browser. These objects include enterprise beans, JavaServer Pages files, and servlets.

The JavaScript debug adapter

which enables server-side JavaScript debugging.

The Compiled language debugger

which allows you to detect and diagnose errors in compiled-language applications.

The Java development tools (JDT) debugger

which allows you to debug Java code.

All of the debug components in the IBM Rational Application Developer for WebSphere can be used for debugging locally and for remote debugging. To learn more about the debug components, launch the IBM Rational Application Developer for WebSphere, select **Help > Help Contents** and choose the **Developing > Debugging applications** bookshelf entries.

Debugging Service details

Use this page to view and modify the settings used by the Debugging Service.

To view this administrative console page, click **Servers > Servers Types > WebSphere application servers > *server name* > Debugging service**.

You can enable a debug session on WebSphere Application Server on this page. Debugging can prove useful when your program behaves differently on the application server than on your local system.

Enable service at server startup

Specifies whether the server will attempt to start the Debug service when the server starts.

JVM debug port

Specifies the port that the Java virtual machine will listen on for debug connections.

JVM debug arguments

Specifies the debugging argument string used to start the JVM in debug mode.

Debug class filters

Specifies an array of classes to ignore during debugging. When running in step-by-step mode, the debugger will not stop in classes that match a filter entry.

Chapter 6. Assembling applications

Application assembly consists of creating Java Platform, Enterprise Edition (Java EE) modules that can be deployed onto application servers. The modules are created from code artifacts such as web application archive (WAR) files, resource adapter archive (RAR) files, enterprise bean (EJB) JAR files, and application client archive (JAR) files. This packaging and configuring of code artifacts into enterprise archive (EAR) modules or stand-alone web modules is necessary for deploying the modules onto an application server.

Before you begin

This topic assumes that you have developed code artifacts that you want to deploy onto an application server and have unit tested the code artifacts in your favorite integrated development environment. Code artifacts that you might assemble into deployable Java EE modules include the following:

- Enterprise beans
- Servlets, JavaServer Pages (JSP) files and other web components
- Resource adapter (connector) implementations
- Client applications
- Session Initiation Protocol (SIP) modules (SAR files)
- Other supporting classes and files

To assemble your code artifacts into deployable Java EE modules, you can use a supported assembly tool. The product supports IBM Rational Application Developer for WebSphere Software for developing, assembling, and deploying Java EE modules.

About this task

You assemble code artifacts into Java EE modules in order to deploy the code artifacts onto an application server. When you assemble code artifacts, you package and configure the code artifacts into deployable Java EE applications and modules, edit annotations or deployment descriptors, and map databases as needed. Unless you assemble your code artifacts into Java EE modules, you cannot run them successfully on an application server.

This topic describes how to assemble Java EE code artifacts into deployable modules using an assembly tool. Alternatively, you can use a rapid deployment tool to quickly assemble and deploy Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4 code artifacts. Refer to “Rapid deployment of J2EE applications” for details.

Procedure

1. Start an assembly tool.
2. Optional: Read the online documentation for the assembly tool.
3. Configure the assembly tool for work on Java EE modules.
4. Migrate J2EE 1.4 or earlier projects or code artifacts created with the Application Server Toolkit, Assembly Toolkit, Application Assembly Tool (AAT) or a different tool.

To migrate files, use the Migration wizard or import the files to the assembly tool.

5. Create an enterprise application project to which you can add archive files. You can create an enterprise application project separately or when you create archive files such as the following:
 - Create a web project.
 - Create an enterprise bean (EJB) project.
 - Create an application client.
 - Create a resource adapter (connector) project.

6. Edit the annotations or deployment descriptors as needed. You can edit annotations or deployment descriptors for enterprise application, Web, application client, resource adapter (connector), and Enterprise JavaBeans (EJB) modules.
Topics in Rational Application Developer documentation provide extensive information on editing annotations or deployment descriptors.
7. Optional: Generate enterprise bean (EJB) to relational database (RDB) mappings for EJB 2.1 or earlier modules.
8. Verify the archive files.
9. Generate code for deployment for web services-enabled modules or for enterprise applications that use web service modules.

What to do next

After assembling your applications, use a systems management tool to deploy the EAR or WAR files onto the application server. “Ways to install enterprise applications or modules” lists systems management tools available for deploying Java EE modules on an application server. The systems management tool follows the security and deployment instructions defined in the annotations or deployment descriptors, and enables you to modify bindings specified within an assembly tool. The tool locates the required external resources that the application uses, such as enterprise beans and databases.

Package your application so that the EAR file contains necessary modules only. Modules can include metadata for the modules such as information on annotations, deployment descriptors, bindings, and IBM extensions.

Use the administrative console at installation to complete the security instructions defined in the annotations or deployment descriptors and to locate required external resources, such as enterprise beans and databases. You can add configuration properties and redefine binding properties defined in an assembly tool.

Application assembly and enterprise applications

Application assembly is the process of creating an enterprise archive (EAR) file containing all files related to an application. This configuration and packaging prepares the application for deployment onto an application server.

EAR files are comprised of the following archives:

- Enterprise bean JAR files (known as EJB modules)
- Web archive (WAR) files (known as web modules)
- Application client JAR files (known as client modules)
- Resource adapter archive (RAR) files (known as resource adapter modules)
- SAR files (known as Session Initiation Protocol (SIP) modules)

Ensure that modules are contained in an EAR file so that they can be deployed onto the server. The exceptions are WAR modules, which you can deploy individually. Although WAR modules can contain regular Java archive (JAR) files, they cannot contain the other module types described previously.

The assembly process includes the following actions:

- Selecting all of the files to include in the module.
- Creating an annotation or deployment descriptor containing instructions for module deployment on the application server.

You can use the graphical interface of Rational Application Developer assembly tools to generate the annotation or deployment descriptor. You can also edit annotations or descriptors directly in your favorite XML editor.

- Packaging modules into a single EAR file, which contains one or more files in a compressed format.

As part of the assembly process, you might also set environment-specific binding information. These bindings are defaults for an administrator to use when installing the application through the administrative console. Further, you might define IBM extensions to the Java Platform, Enterprise Edition (Java EE) specifications, such as to allow servlets to be served by class name. To ensure portability to other application servers, these extensions are saved in an XML file that is separate from the standard annotation or deployment descriptor.

Restriction: Do not include a pound sign (#) in the name of files that are packaged within an application archive. Due to internal processing, the application server fails to correctly deploy the application when a pound sign is included in a file name within the application archive. When this failure occurs, an exception might occur when the application is being processed. Also, parts of the application might be missing after the application is deployed. To address this issue, rename any file names within the application archive so that they do not contain a pound sign.

Development and assembly tools

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

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

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

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

IBM WebSphere Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java EE applications to WebSphere Application Server Version 7.0 and 8.x. This workbench integrates with the application server to help you to quickly deploy and test applications. This product contains wizards and visual editors that support the Java EE programming model.

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

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

Generating code for web service deployment

Before deploying web services-enabled modules or any enterprise application archive (EAR) files that contain web services-enabled module onto an application server, you must generate deployment code for the application.

Before you begin

This topic assumes you have assembled a module enabled with web services, added it to an application, saved the application, and verified the application. It also assumes that you have started and configured an assembly tool.

About this task

You can use an assembly tool to generate deployment code for the web services-enabled module or for the EAR file that contains the web services-enabled module.

Procedure

1. If you have turned automatic validation off, manually validate any modules that use web services with the JSR109 web services validator before generating deployment code for them. If validating your module results in compilation errors or validation errors, fix the errors before generating deployment code. However, if validating your module results in warning or information messages, you can generate deployment code.
2. In the Project Explorer view of the assembly tool, right-click on the web services-enabled module (WAR, enterprise bean JAR, or application client JAR file) for which you want to generate code for deployment.
3. Click **Deploy**. Alternatively, you can generate deployment code for web services-enabled modules using the deployment tool for web services (**wsdeploy**) from a command prompt.
4. If messages indicate that automatic file overwriting is not enabled, click **Yes to All** so the generated files are added to the module.

Results

Code is generated into the folder where your web services-enable module is located. Problems with the generation of code result in a window that displays error messages.

What to do next

Install the Java Platform, Enterprise Edition (Java EE) application on your server machine. You can install the application onto a server using the administrative console. Before installing the application, you might need to set class paths.

Assembling applications: Resources for learning

Additional information and guidance on assembling applications is available on various Internet sites.

Use the following links to find relevant supplemental information about the application assembly and using an assembly tool. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

View links to additional information about:

- “Programming instructions and examples” on page 19
- “Programming specifications” on page 19
- “Administration” on page 19

Programming instructions and examples

- Rational Application Developer V8 Programming Guide, SG24-7835-00, <http://www.redbooks.ibm.com/abstracts/sg247835.html?open>
- Rational developer community, <http://www.ibm.com/developerworks/rational/>
- *IBM WebSphere Developer Technical Journal: Using Rational Developer to create a simple web service and use it in a web application*, http://www.ibm.com/developerworks/websphere/techjournal/0506_parkin/0506_parkin.html
- Java EE Tutorials, <http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html>
- Recommended reading list: Java EE and WebSphere Application Server, http://www.ibm.com/developerworks/websphere/library/techarticles/0305_issw/recommendedreading.html

Programming specifications

- Specifications and API documentation

Administration

- WebSphere Application Server V7 Administration and Configuration Guide, SG24-7615-01, <http://www.redbooks.ibm.com/abstracts/sg247615.html>

Chapter 7. Class loading

Class loaders are part of the Java virtual machine (JVM) code and are responsible for finding and loading class files. Class loaders enable applications that are deployed on servers to access repositories of available classes and resources. Application developers and deployers must consider the location of class and resource files, and the class loaders used to access those files, to make the files available to deployed applications. Class loaders affect the packaging of applications and the runtime behavior of packaged applications of deployed applications.

Before you begin

This topic describes how to configure class loaders for application files or modules that are installed on an application server.

To better understand class loaders in WebSphere Application Server, read “Class loaders.” The topic “Class loading: Resources for learning” on page 31 refers to additional sources.

About this task

Configure class loaders for application files or modules that are installed on an application server using the administrative console. You configure class loaders to ensure that deployed application files and modules can access the classes and resources that they need to run successfully.

Procedure

1. If an installed application module uses a resource, create a resource provider that specifies the directory name of the resource drivers.
Do not specify the resource Java archive (JAR) file names. All JAR files in the specified directory are added into the class path of the WebSphere Application Server extensions class loader. If a resource driver requires a native library (.dll or .so file), specify the name of the directory that contains the library in the native path of the resource configuration.
2. Specify class-loader values for an application server.
3. Specify class-loader values for an installed enterprise application.
4. Specify the class-loader mode for an installed web module.
5. If your deployed application uses shared library files, associate the shared library files with your application. Use a library reference to associate a shared library file with your application.
 - a. If you have not done so already, define shared libraries for library files that your applications need.
 - b. Define a library reference for each shared library that your application uses.

What to do next

After configuring class loaders, ensure that your application performs as desired. To diagnose and fix problems with class loaders, refer to Troubleshooting class loaders.

Class loaders

Class loaders find and load class files. Class loaders enable applications that are deployed on servers to access repositories of available classes and resources. Application developers and deployers must consider the location of class and resource files, and the class loaders used to access those files, to make the files available to deployed applications.

This topic provides the following information about class loaders in WebSphere Application Server:

- “Class loaders used and the order of use” on page 22
- “Class-loader isolation policies” on page 23

- “Class-loader modes” on page 25

Class loaders used and the order of use

The product runtime environment uses the following class loaders to find and load new classes for an application in the following order:

1. The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine

The bootstrap class loader uses the boot class path (typically classes in `jre/lib`) to find and load classes. The extensions class loader uses the system property `java.ext.dirs` (typically `jre/lib/ext`) to find and load classes. The CLASSPATH class loader uses the CLASSPATH environment variable to find and load classes.

The CLASSPATH class loader loads the Java Platform, Enterprise Edition (Java EE) application programming interfaces (APIs) provided by the WebSphere Application Server product in the `j2ee.jar` file. Because this class loader loads the Java EE APIs, you can add libraries that depend on the Java EE APIs to the class path system property to extend a server class path. However, a preferred method of extending a server class path is to add a shared library.

2. A WebSphere extensions class loader

The WebSphere extensions class loader loads the WebSphere Application Server classes that are required at run time. The extensions class loader uses a `ws.ext.dirs` system property to determine the path that is used to load classes. Each directory in the `ws.ext.dirs` class path and every Java archive (JAR) file or compressed file in these directories is added to the class path used by this class loader.

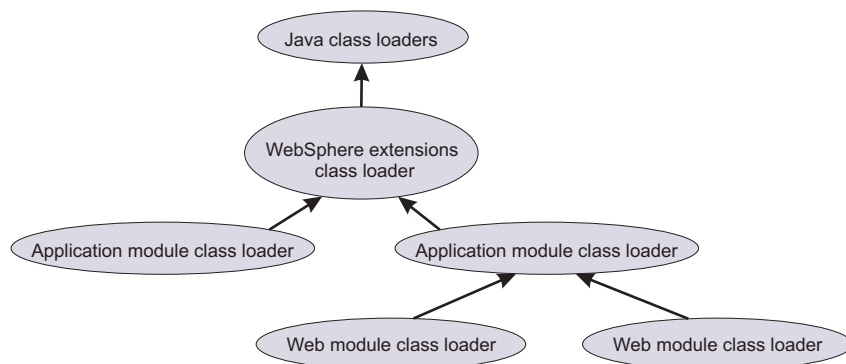
The WebSphere extensions class loader also loads resource provider classes into a server if an application module installed on the server refers to a resource that is associated with the provider and if the provider specifies the directory name of the resource drivers.

3. One or more application module class loaders that load elements of enterprise applications running in the server

The application elements can be web modules, enterprise bean (EJB) modules, resource adapter archives (RAR files), and dependency JAR files. Application class loaders follow Java EE class-loading rules to load classes and JAR files from an enterprise application. The product enables you to associate shared libraries with an application.

4. Zero or more web module class loaders

By default, web module class loaders load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. Web module class loaders are children of application class loaders. You can specify that an application class loader load the contents of a web module rather than the web module class loader.



Each class loader is a child of the previous class loader. That is, the application module class loaders are children of the WebSphere extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its parent class loader. If none of the parent class loaders can find the class, the original class loader attempts to load the class. Requests can only go to a parent class loader; they cannot go to a child class loader. If the WebSphere extensions class loader is requested to find a class in a Java EE module, it cannot go to the

application module class loader to find that class and a `ClassNotFoundException` error occurs. After a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader or go up the precedence list until the class is found.

Class-loader isolation policies

The number and function of the application module class loaders depend on the class-loader policies that are specified in the server configuration. Class loaders provide multiple options for isolating applications and modules to enable different application packaging schemes to run on an application server.

Two class-loader policies control the isolation of applications and modules:

Table 1. Class-loader policy descriptions. Available policies include Application and WAR.

Class-loader policy	Description
Application	Application class loaders load EJB modules, dependency JAR files, embedded resource adapters, and application-scoped shared libraries. Depending on the application class-loader policy, an application class loader can be shared by multiple applications (<code>Single</code>) or unique for each application (<code>Multiple</code>). The application class-loader policy controls the isolation of applications that are running in the system. When set to <code>Single</code> , applications are not isolated. When set to <code>Multiple</code> , applications are isolated from each other.
WAR	<p>By default, web module class loaders load the contents of the <code>WEB-INF/classes</code> and <code>WEB-INF/lib</code> directories. The application class loader is the parent of the web module class loader. You can change the default behavior by changing the web application archive (WAR) class-loader policy of the application.</p> <p>The WAR class-loader policy controls the isolation of web modules. If this policy is set to <code>Application</code>, then the Web module contents also are loaded by the application class loader (in addition to the EJB files, RAR files, dependency JAR files, and shared libraries). If the policy is set to <code>Module</code>, then each web module receives its own class loader whose parent is the application class loader.</p> <p>Tip: The console and the underlying <code>deployment.xml</code> file use different names for WAR class-loader policy values. In the console, the WAR class-loader policy values are <code>Application</code> or <code>Module</code>. However, in the underlying <code>deployment.xml</code> file where the policy is set, the WAR class-loader policy values are <code>Single</code> instead of <code>Application</code>, or <code>Multiple</code> instead of <code>Module</code>. <code>Application</code> is the same as <code>Single</code>, and <code>Module</code> is the same as <code>Multiple</code>.</p>

Restriction: WebSphere Application Server class loaders never load application client modules.

For each application server in the system, you can set the application class-loader policy to `Single` or `Multiple`. When the application class-loader policy is set to `Single`, then a single application class loader loads all EJB modules, dependency JAR files, and shared libraries in the system. When the application class-loader policy is set to `Multiple`, then each application receives its own class loader that is used for loading the EJB modules, dependency JAR files, and shared libraries for that application.

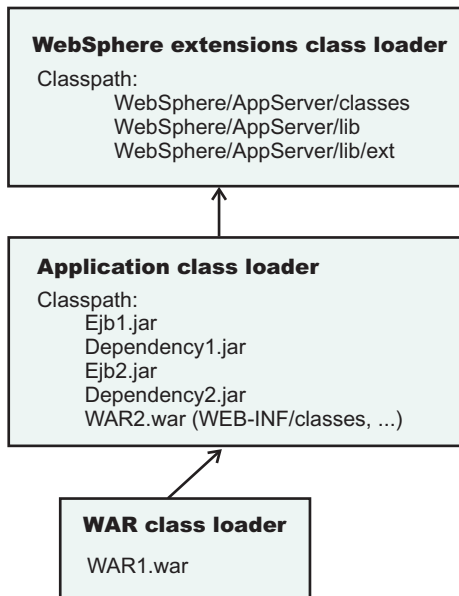
An application class loader loads classes from web modules if the application's WAR class-loader policy is set to `Application`. If the application's WAR class-loader policy is set to `Module`, then each WAR module receives its own class loader.

The following example shows that when the application class-loader policy is set to `Single`, a single application class loader loads all of the EJB modules, dependency JAR files, and shared libraries of all applications on the server. The single application class loader can also load web modules if an application has its WAR class-loader policy set to `Application`. Applications that have a WAR class-loader policy set to `Module` use a separate class loader for web modules.

Server's application class-loader policy: Single
Application's WAR class-loader policy: Module

Application 1
Module: EJB1.jar
Module: WAR1.war
MANIFEST Class-Path: Dependency1.jar
WAR Classloader Policy = Module

Application 2
Module: EJB2.jar
MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
WAR Classloader Policy = Application

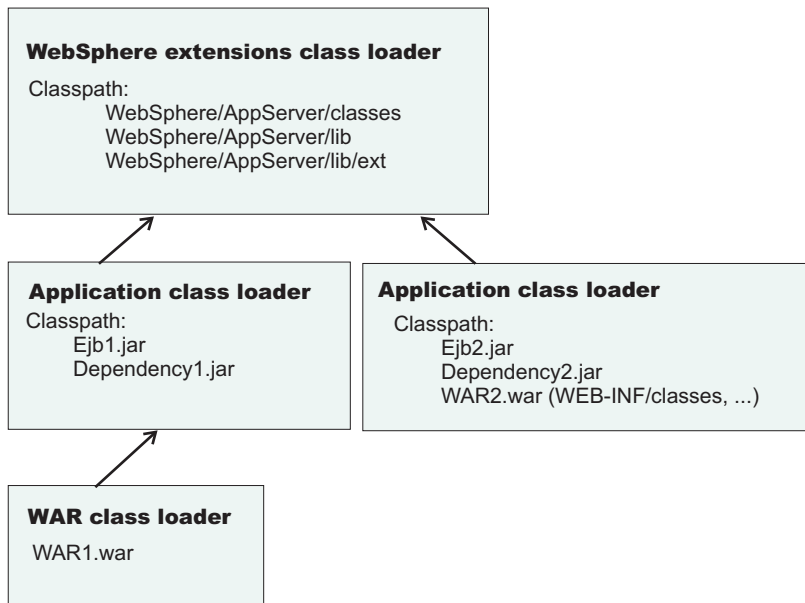


The following example shows that when the application class-loader policy of an application server is set to Multiple, each application on the server has its own class loader. An application class loader also loads its web modules if the application WAR class-loader policy is set to Application. If the policy is set to Module, then a web module uses its own class loader.

Server's application class-loader policy: Multiple
Application's WAR class-loader policy: Module

Application 1
Module: EJB1.jar
Module: WAR1.war
MANIFEST Class-Path: Dependency1.jar
WAR Classloader Policy = Module

Application 2
Module: EJB2.jar
MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
WAR Classloader Policy = Application



Class-loader modes

The class-loader delegation mode, also known as the *class loader order*, determines whether a class loader delegates the loading of classes to the parent class loader. The following values for class-loader mode are supported:

Table 2. Class-loader mode descriptions. Available modes include *Parent first* and *Parent last*.

Class-loader mode	Description
Parent first Also known as Classes loaded with parent class loader first .	The Parent first class-loader mode causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. This value is the default for the class-loader policy and for standard JVM class loaders.
Parent last Also known as Classes loaded with local class loader first or Application first .	The Parent last class-loader mode causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Using this policy, an application class loader can override and provide its own version of a class that exists in the parent class loader.

The following settings determine the mode of a class loader:

- If the application class-loader policy of an application server is `Single`, the server-level mode value defines the mode for an application class loader.
- If the application class-loader policy of an application server is `Multiple`, the application-level mode value defines the mode for an application class loader.
- If the WAR class-loader policy of an application is `Module`, the module-level mode value defines the mode for a WAR class loader.

Configuring class loaders of a server

You can configure the application class loaders for an application server. Class loaders enable applications that are deployed on the application server to access repositories of available classes and resources.

Before you begin

This topic assumes that an administrator created an application server on a WebSphere Application Server product.

About this task

Configure the class loaders of an application server to set class-loader policy and mode values which affect all applications that are deployed on the server. Use the administrative console to configure the class loaders.

Procedure

1. Click **Servers > Server Types > WebSphere application servers > *server_name*** to access an application server settings page.
2. Specify the application class-loader policy for the application server.

The application class-loader policy controls the isolation of applications that run in the system (on the server). An application class loader groups enterprise bean (EJB) modules, shared libraries, resource adapter archives (RAR files), and dependency Java archive (JAR) files associated to an application. Dependency JAR files are JAR files that contain code which can be used by both enterprise beans and servlets. The application class-loader policy controls whether an application class loader can be shared by multiple applications or is unique for each application.

Use the application server settings page to specify the application class-loader policy for the server:

Option	Description
Single	Applications are not isolated from each other. Uses a single application class loader to load all of the EJB modules, shared libraries, and dependency JAR files in the system.
Multiple	Applications are isolated from each other. Gives each application its own class loader to load the EJB modules, shared libraries, and dependency JAR files of that application.

3. Specify the application class-loader mode for the application server.
The application class loading mode specifies the class-loader mode when the application class-loader policy is *Single*.

On the application server settings page, select either of the following values:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. Classes loaded with parent class loader first is the default value for class loading mode. This value is also known as parent first.
Classes loaded with local class loader first (parent last)	Causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Using this policy, an application class loader can override and provide its own version of a class that exists in the parent class loader.

4. Specify the class-loader mode for the class loader.
 - a. On the application server settings page, click **Java and Process Management > Class loader** to access the Class loader page.

- b. On the Class loader page, click **New** to access the settings page for a class loader.
- c. On the class loader settings page, specify the class loader order.

The Classes loaded with parent class loader first value causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path.

The Classes loaded with local class loader first (parent last) value causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent.

- d. Click **OK**.

An identifier is assigned to a class-loader instance. The instance is added to the collection of class loaders shown on the Class loader page.

What to do next

Save the changes to the administrative configuration.

Class loader collection

Use this page to manage class-loader instances on an application server. A class loader determines whether an application class loader or a parent class loader finds and loads Java class files for an application.

To view this administrative console page, click **Servers > Server Types > WebSphere application servers > *server_name* > Java and Process Management > Class loader**.

Class loader ID

Specifies a string that is unique to the server identifying the class-loader instance. The product assigns the identifier.

Class loader order

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

Class loader settings

Use this page to configure a class loader for applications that reside on an application server.

To view this administrative console page, click **Servers > Server Types > WebSphere application servers > *server_name* > Java and Process Management > Class loader > *class_loader_ID***.

Class loader ID

Specifies a string that is unique to the server identifying the class-loader instance. The product assigns the identifier.

Information	Value
Data type	String

Class loader order

Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for development kit class loaders and WebSphere Application Server

class loaders is Classes loaded with parent class loader first. By specifying Classes loaded with local class loader first (parent last), your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overridden classes and non-overridden classes.

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

For your application to use the default configuration of Jakarta Commons Logging in this product, set this application class loader order to Classes loaded with parent class loader first. For your application to override the default configuration of Jakarta Commons Logging, your application must provide the configuration in a form supported by Jakarta Commons Logging and this class loader order must be set to Classes loaded with local class loader first (parent last). Also, to override the default configuration, set the class loader order for each web module in your application so that the correct logger factory loads.

Information	Value
Data type	String
Default	Parent first

Configuring application class loaders

You can set values that control the class-loading behavior of an installed enterprise application. Class loaders enable an application to access repositories of available classes and resources.

Before you begin

This topic assumes that you installed an application on an application server.

About this task

Configure the class loaders of an enterprise application to set class-loader policy and mode values for this application.

An application class loader groups enterprise bean (EJB) modules, shared libraries, resource adapter archive (RAR) files, and dependency Java archive (JAR) files associated to an application. Dependency JAR files are JAR files that contain code which can be used by both enterprise beans and servlets.

An application class loader is the parent of a web application archive (WAR) class loader. By default, a web module has its own WAR class loader to load the contents of the web module. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the Web module.

Use the administrative console to configure the class loaders.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Procedure

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Class loading and update detection** to access the settings page for an application class loader.

- Specify whether to reload application classes when the application or its files are updated.
By default, class reloading is not enabled. Select **Override class reloading settings for web and EJB modules** to choose to reload application classes. You might specify different values for EJB modules and for web modules such as servlets and JavaServer Pages (JSP) files.

- Specify the number of seconds to scan the application's file system for updated files.

The value specified for **Polling interval for updated files** takes effect only if class reloading is enabled. The default is the value of the reloading interval attribute in the IBM extension (META-INF/ibm-application-ext.xml) file of the enterprise application (EAR file). You might specify different values for EJB modules and for web modules such as servlets and JSP files.

To enable reloading, specify an integer value that is greater than zero (for example, 1 to 2147483647). To disable reloading, specify zero (0).

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The ibm-webservices-ext.xmi, ibm-webservices-bnd.xml, ibm-webservicesclient-bnd.xml, ibm-webservicesclient-ext.xmi, and ibm-portlet-ext.xmi files continue to use the .xmi file extensions.

- Specify the class loader order for the application.

The application class loader order specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The default is to search in the parent class loader before searching in the application class loader to load a class.

Select either of the following values for **Classes loader order**:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to search in the parent class loader first to load a class. This value is the standard for Development Kit class loaders and WebSphere Application Server class loaders.
Classes loaded with local class loader first (parent last)	Causes the class loader to search in the application class loader first to load a class. By specifying Classes loaded with local class loader first (parent last) , your application can override classes contained in the parent class loader. Note: Specifying the Classes loaded with local class loader first (parent last) value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

- Specify whether to use a single or multiple class loaders to load web application archives (WAR files) of your application.

By default, web modules have their own WAR class loader to load the contents of the WEB-INF/classes and WEB-INF/lib directories. The default WAR class loader value is **Class loader** for each WAR file

in application, which uses a separate class loader to load each WAR file. Setting the value to `Single class loader for application` causes the application class loader to load the web module contents as well as the EJB modules, shared libraries, RAR files, and dependency JAR files associated to the application. The application class loader is the parent of the WAR class loader.

Select either of the following values for **WAR class loader policy**:

Option	Description
Class loader for each WAR file in application	Uses a different class loader for each WAR file.
Single class loader for application	Uses a single class loader to load all of the WAR files in your application.

6. Click **OK**.

What to do next

Save the changes to the administrative configuration.

Configuring web module class loaders

You can set values that control the class-loading behavior of an installed web module.

Before you begin

This topic assumes that you installed a web module on an application server.

About this task

Configure the class loader order value of an installed web module. By default, a web module has its own web application archive (WAR) class loader to load the contents of the web module, which are in the `WEB-INF/classes` and `WEB-INF/lib` directories.

An application class loader is the parent of a WAR class loader. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the web module.

The default WAR class loader policy value is `Class loader for each WAR file in application`. If the policy is set to `Class loader for each WAR file in application`, then each web module receives its own class loader whose parent is the application class loader. If the policy is set to `Single class loader for application`, then the application class loader loads the web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter archive (RAR) files, and dependency Java archive (JAR) files associated to an application. Thus, the configuration of the parent application class loader affects the WAR class loader. You can set the policy on the Class loading and update detection page of an administrative console.

Use the administrative console to configure the application and WAR class loaders.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Procedure

1. If you have not done so already, configure the application class loader.

Settings such as **Override class reloading settings for web and EJB modules**, **Polling interval for updated files** and **WAR class loader policy** can affect web module class loading.

If **WAR class loader policy** is set to `Class loader` for each WAR file in application, then the web module receives its own class loader and the WAR class-loader policy of the web module defines the mode for a WAR class loader. If the policy is set to `Single class loader` for application, then the application class loader loads the web module contents.

2. Specify the class loader order for the installed web module.

The web module class-loader mode specifies whether the class loader searches in the parent application class loader or in the WAR class loader first to load a class. The default is to search in the parent application class loader before searching in the WAR class loader to load a class.

Select either of the following values for **Class loader order**:

Option	Description
Classes loaded with parent class loader first	This option causes the class loader to <i>prefer classes that are provided by the product</i> over the classes that exist within the web module. This approach is standard for Development Kit class loaders and WebSphere Application Server class loaders.
Classes loaded with local class loader first	This option causes the class loader to <i>prefer classes that exist in the web module</i> over the classes that are provided by the product. If the same class exists in both the product and the web module, the class from the web module is loaded. Attention: If you specify the <code>Classes loaded with local class loader first</code> value, you might receive <code>LinkageErrors</code> or <code>ClassCastException</code> messages if you have mixed use of overridden classes and non-overridden classes.

3. Click **OK**.

What to do next

Save the changes to the administrative configuration.

Class loading: Resources for learning

Additional information and guidance on class loading is available on various Internet sites.

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

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

View links to additional information about:

- “Programming model and decisions” on page 32
- “Programming instructions and examples” on page 32
- “Programming specifications” on page 32

Programming model and decisions

- Demystifying class loading problems, Part 1: An introduction to class loading and debugging tools - Learn how class loading works and how your JVM can help you sort out class loading problems (*developerWorks*, November 2005), http://www.ibm.com/developerworks/java/library/j-dclp1/?S_TACT=106AH10W&S_CMP=NC
- Demystifying class loading problems, Part 2: Basic class loading exceptions - An in-depth look at some simple class loading quirks and conundrums (*developerWorks*, December 2005), http://www.ibm.com/developerworks/java/library/j-dclp2.html?S_TACT=105AGX10&S_CMP=NC
- Demystifying class loading problems, Part 3: Tackling more unusual class loading problems - Understand class loading and quash subtle exceptions (*developerWorks*, December 2005), http://www.ibm.com/developerworks/java/library/j-dclp3/?S_TACT=105AGX10&S_CMP=NC
- J2EE Class Loading Demystified (*developerWorks*, August 2002), http://www.ibm.com/developerworks/websphere/library/techarticles/0112_deboer/deboer.html
- Java programming dynamics, Part 1: Classes and class loading - A look at classes and what goes on as they're loaded by a JVM (*developerWorks*, April 2003), <http://www.ibm.com/developerworks/java/library/j-dyn0429/>

Programming instructions and examples

- WebSphere Application Server V7: Understanding Class Loaders, <http://publib-b.boulder.ibm.com/abstracts/redp4581.html?0pen>

Programming specifications

- Specifications and API documentation

Chapter 8. Adding logging and tracing to your application

You can add logging and tracing to applications to help analyze performance and diagnose problems in WebSphere Application Server.

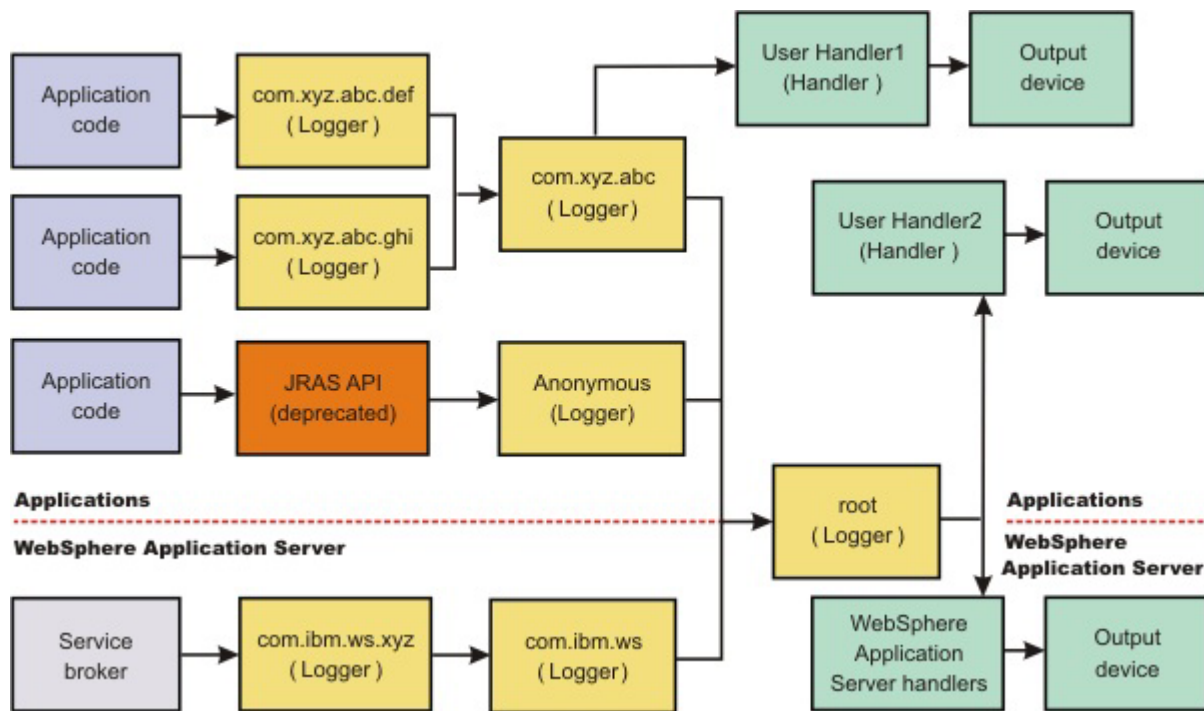
About this task

Deprecation: The JRAs framework that is described in this information center is deprecated. However, you can achieve the same results using Java logging.

Designers and developers of applications that run with or under WebSphere Application Server, such as servlets, JavaServer Pages (JSP) files, enterprise beans, client applications, and their supporting classes, might find it useful to use Java logging for generating their application logging.

This approach has advantages over adding `System.out.println` statements to your code:

- Your messages are displayed in the WebSphere Application Server standard log files, using a standard message format with additional data, such as a date and time stamp that are added automatically.
- You can more easily correlate problems and events in your own application to problems and events that are associated with WebSphere Application Server components.
- You can take advantage of the WebSphere Application Server log file management features.



Procedure

1. Enable and configure any of the supported types of logging as needed. Use one of the following methods:
 - Configuring Java logging using the administrative console
 - Configuring applications to use Jakarta Commons Logging
2. Customize the properties to meet your logging needs. For example, enable or disable a particular log, specify the number of logs to be kept, and specify a format for log output.
 - Configuring Java logging using the administrative console

3. If you do not want log and trace from Jakarta Commons Logging to use the WebSphere log and trace infrastructure, reconfigure the Jakarta Commons Logging.
 - “Configuring applications to use Jakarta Commons Logging” on page 49

Note: Use the WebSphere log and trace infrastructure for all of your log content to make problem source identification simpler.
4. Restart the application server after making static configuration changes.

Example

The sample security policy that follows grants access to the file system and runtime classes. Include this security policy, with the entry permission `java.util.logging.LoggingPermission "control"`, in the META-INF directory of your application if you want your applications to programmatically alter controlled properties of loggers and handlers. The META-INF file is located in the following locations for the different module types:

Project name	Location
EJB projects	ejbModule/META-INF/MANIFEST.MF
Application client projects	appClientModule/META-INF/MANIFEST.MF
Dynamic web projects	WebContent/META-INF/MANIFEST.MF
Connector projects	connectorModule/META-INF/MANIFEST.MF

Below is a sample security policy that grants permission to modify logging properties:

```

////////////////////////////////////
//
// WebSphere Application Server Security Policy
//
////////////////////////////////////

////////////////////////////////////
// Allow all access to the file system and runtime classes
////////////////////////////////////
grant codeBase "file:${application}" {
    permission java.util.logging.LoggingPermission "control";
};

```

Using Java logging in an application

This topic describes how to use Java logging within an application.

About this task

To create an application using Java logging, perform the following steps:

Procedure

1. Optional: Create the necessary handler, formatter, and filter classes if you need your own log files.

Note: Use the WebSphere log and trace infrastructure to make problem source identification simpler, rather than creating separate log files.
2. Optional: If localized messages are used by the application, create a resource bundle, as described in “Creating log resource bundles and message files” on page 46.
3. In the application code, get a reference to a logger instance, as described in “Using a logger” on page 35.

4. Insert the appropriate message and trace logging statements in the application, as described in “Using a logger.”

Using a logger

You can use Java logging to log messages and add tracing.

About this task

Java provides a log and trace package, `java.util.logging`, that you can use to instrument your applications. This topic provides recommendations about how to use the log and trace package.

Procedure

1. Use `WsLevel.DETAIL` level and above for messages, and lower levels for trace. The WebSphere Application Server Extension API (the `com.ibm.websphere.logging` package) contains the `WsLevel` class.

For messages use:

```
WsLevel.FATAL
Level.SEVERE
Level.WARNING
WsLevel.AUDIT
Level.INFO
Level.CONFIG
WsLevel.DETAIL
```

For trace use:

```
Level.FINE
Level.FINER
Level.FINEST
```

2. Use the `logp` method instead of the `log` or the `logrb` method. The `logp` method accepts parameters for class name and method name. The `log` and `logrb` methods will generally try to infer this information, but the performance penalty is prohibitive. In general, the `logp` method has less performance impact than the `log` or the `logrb` method.
3. Avoid using the `logrb` method. This method leads to inefficient caching of resource bundles and poor performance.
4. Use the `isLoggable` method to avoid creating data for a logging call that does not get logged. For example:

```
if (logger.isLoggable(Level.FINEST)) {
    String s = dumpComponentState(); // some expensive to compute method
    logger.logp(Level.FINEST, className, methodName, "componentX state
dump:\n{0}", s);
}
```

Localized messages

The following sample applies to localized messages:

```
// note - generally avoid use of FINE, FINER, FINEST levels for messages to be consistent with
// WebSphere Application Server

String componentName = "com.ibm.websphere.componentX";
String resourceBundleName = "com.ibm.websphere.componentX.Messages";
Logger logger = Logger.getLogger(componentName, resourceBundleName);

// "Convenience" methods - not generally recommended due to lack of class
// method names
// - cannot specify message substitution parameters
// - cannot specify class and method names
if (logger.isLoggable(Level.SEVERE))
    logger.severe("MSG_KEY_01");

if (logger.isLoggable(Level.WARNING))
    logger.warning("MSG_KEY_01");

if (logger.isLoggable(Level.INFO))
    logger.info("MSG_KEY_01");
```

```

if (logger.isLoggable(Level.CONFIG))
    logger.config("MSG_KEY_01");

// log methods are not generally used due to lack of class and method
names
// - enable use of WebSphere Application Server-specific levels
// - enable use of message substitution parameters
// - cannot specify class and method names
if (logger.isLoggable(WsLevel.FATAL))
    logger.log(WsLevel.FATAL, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.SEVERE))
    logger.log(Level.SEVERE, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.WARNING))
    logger.log(Level.WARNING, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.AUDIT))
    logger.log(WsLevel.AUDIT, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.INFO))
    logger.log(Level.INFO, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.CONFIG))
    logger.log(Level.CONFIG, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.DETAIL))
    logger.log(WsLevel.DETAIL, "MSG_KEY_01", "parameter 1");

// logp methods are the way to log
// - enable use of WebSphere Application Server-specific levels
// - enable use of message substitution parameters
// - enable use of class and method names
if (logger.isLoggable(WsLevel.FATAL))
    logger.logp(WsLevel.FATAL, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.SEVERE))
    logger.logp(Level.SEVERE, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.WARNING))
    logger.logp(Level.WARNING, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(WsLevel.AUDIT))
    logger.logp(WsLevel.AUDIT, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.INFO))
    logger.logp(Level.INFO, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.CONFIG))
    logger.logp(Level.CONFIG, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(WsLevel.DETAIL))
    logger.logp(WsLevel.DETAIL, className, methodName, "MSG_KEY_01",
"parameter 1");

// logrb methods are not generally used due to diminished performance
of switching resource bundles dynamically
// - enable use of WebSphere Application Server-specific levels
// - enable use of message substitution parameters
// - enable use of class and method names
String resourceBundleNameSpecial =
"com.ibm.websphere.componentX.MessagesSpecial";

if (logger.isLoggable(WsLevel.FATAL))
    logger.logrb(WsLevel.FATAL, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.SEVERE))
    logger.logrb(Level.SEVERE, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.WARNING))
    logger.logrb(Level.WARNING, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.AUDIT))
    logger.logrb(WsLevel.AUDIT, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.INFO))
    logger.logrb(Level.INFO, className, methodName, resourceBundleNameSpecial,

```

```

"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.CONFIG))
    logger.logrb(Level.CONFIG, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WebSocketLevel.DETAIL))
    logger.logrb(WebSocketLevel.DETAIL, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

```

For trace, or content that is not localized, the following sample applies:

```

// note - generally avoid use of FATAL, SEVERE, WARNING, AUDIT,
// INFO, CONFIG, DETAIL levels for trace
// to be consistent with WebSphere Application Server

String componentName = "com.ibm.websphere.componentX";
Logger logger = Logger.getLogger(componentName);

// Entering / Exiting methods are used for non trivial methods
if (logger.isLoggable(Level.FINER))
    logger.entering(className, methodName);

if (logger.isLoggable(Level.FINER))
    logger.entering(className, methodName, "method param1");

if (logger.isLoggable(Level.FINER))
    logger.exiting(className, methodName);

if (logger.isLoggable(Level.FINER))
    logger.exiting(className, methodName, "method result");

// Throwing method is not generally used due to lack of message - use
// logp with a throwable parameter instead
if (logger.isLoggable(Level.FINER))
    logger.throwing(className, methodName, throwable);

// Convenience methods are not generally used due to lack of class
// method names
// - cannot specify message substitution parameters
// - cannot specify class and method names
if (logger.isLoggable(Level.FINE))
    logger.fine("This is my trace");

if (logger.isLoggable(Level.FINER))
    logger.finer("This is my trace");

if (logger.isLoggable(Level.FINEST))
    logger.finest("This is my trace");

// log methods are not generally used due to lack of class and
// method names
// - enable use of WebSphere Application Server-specific levels
// - enable use of message substitution parameters
// - cannot specify class and method names
if (logger.isLoggable(Level.FINE))
    logger.log(Level.FINE, "This is my trace", "parameter 1");

if (logger.isLoggable(Level.FINER))
    logger.log(Level.FINER, "This is my trace", "parameter 1");

if (logger.isLoggable(Level.FINEST))
    logger.log(Level.FINEST, "This is my trace", "parameter 1");

// logp methods are the recommended way to log
// - enable use of WebSphere Application Server-specific levels
// - enable use of message substitution parameters
// - enable use of class and method names
if (logger.isLoggable(Level.FINE))
    logger.logp(Level.FINE, className, methodName, "This is my trace",
"parameter 1");

if (logger.isLoggable(Level.FINER))
    logger.logp(Level.FINER, className, methodName, "This is my trace",
"parameter 1");

if (logger.isLoggable(Level.FINEST))
    logger.logp(Level.FINEST, className, methodName, "This is my trace",
"parameter 1");

// logrb methods are not applicable for trace logging because no localization
// is involved

```

There may be occasions when you want to propagate log records to your own log handlers rather than participate in integrated logging. To use a stand-alone log handler, set the `useParentHandlers` flag to `false` in your application. The mechanism for creating a custom handler is the Handler class support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with handlers, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the `java.util.logging` API. The following sample shows a custom handler:

```
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.logging.Handler;
import java.util.logging.LogRecord;

/**
 * MyCustomHandler outputs contents to a specified file
 */
public class MyCustomHandler extends Handler {

    FileOutputStream fileOutputStream;
    PrintWriter printWriter;

    public MyCustomHandler(String filename) {
        super();

        // check input parameter
        if (filename == null || filename == "")
            filename = "mylogfile.txt";

        try {
            // initialize the file
            fileOutputStream = new FileOutputStream(filename);
            printWriter = new PrintWriter(fileOutputStream);
            setFormatter(new SimpleFormatter());
        }
        catch (Exception e) {
            // implement exception handling...
        }
    }

    /* (non-API documentation)
     * @see java.util.logging.Handler#publish(java.util.logging.LogRecord)
     */
    public void publish(LogRecord record) {
        // ensure that this log record should be logged by this Handler
        if (!isLoggable(record))
            return;

        // Output the formatted data to the file
        printWriter.println(getFormatter().format(record));
    }

    /* (non-API documentation)
     * @see java.util.logging.Handler#flush()
     */
    public void flush() {
        printWriter.flush();
    }

    /* (non-API documentation)
     * @see java.util.logging.Handler#close()
     */
    public void close() throws SecurityException {
        printWriter.close();
    }
}
```


A custom filter provides optional, secondary control over what is logged, beyond the control that is provided by the level. The mechanism for creating a custom filter is the Filter interface support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with filters, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation the for the java.util.logging API.

The following example shows a custom filter:

```
/**
 * This class filters out all log messages starting with SECJ022E, SECJ0373E, or SECJ0350E.
 */
import java.util.logging.Filter;
import java.util.logging.Handler;
import java.util.logging.Logger;
import java.util.logging.LogRecord;

public class MyFilter implements Filter {
    public boolean isLoggable(LogRecord lr) {
        String msg = lr.getMessage();
        if (msg.startsWith("SECJ022E") || msg.startsWith("SECJ0373E") || msg.startsWith("SECJ0350E")) {
            return false;
        }
        return true;
    }
}

//This code will register the above log filter with the root Logger's handlers (including the WAS system logs):
...
Logger rootLogger = Logger.getLogger("");
rootLogger.setFilter(new MyFilter());
```

A formatter formats events. Handlers are associated with one or more formatters. The mechanism for creating a custom formatter is the Formatter class support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with formatters, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.logging API.

The following example shows a custom formatter:

```
import java.util.Date;
import java.util.logging.Formatter;
import java.util.logging.LogRecord;

/**
 * MyCustomFormatter formats the LogRecord as follows:
 * date level localized message with parameters
 */
public class MyCustomFormatter extends Formatter {

    public MyCustomFormatter() {
        super();
    }

    public String format(LogRecord record) {

        // Create a StringBuffer to contain the formatted record
        // start with the date.
        StringBuffer sb = new StringBuffer();

        // Get the date from the LogRecord and add it to the buffer
        Date date = new Date(record.getMillis());
        sb.append(date.toString());
        sb.append(" ");

        // Get the level name and add it to the buffer
        sb.append(record.getLevel().getName());
        sb.append(" ");

        // Get the formatted message (includes localization
        // and substitution of paramters) and add it to the buffer
        sb.append(formatMessage(record));
        sb.append("\n");
    }
}
```

```

    return sb.toString();
}
}

```

Adding custom handlers, filters, and formatters enables you to customize your logging environment beyond what can be achieved by the configuration of the default WebSphere Application Server logging infrastructure. The following example demonstrates how to add a new handler to process requests to the `com.myCompany` subtree of loggers (see “Configuring the logger hierarchy” on page 45). The main method in this sample gives an example of how to use the newly configured logger.

```

import java.util.Vector;
import java.util.logging.Filter;
import java.util.logging.Formatter;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MyCustomLogging {

    public MyCustomLogging() {
        super();
    }

    public static void initializeLogging() {

        // Get the logger that you want to attach a custom Handler to
        String defaultResourceBundleName = "com.myCompany.Messages";
        Logger logger = Logger.getLogger("com.myCompany", defaultResourceBundleName);

        // Set up a custom Handler (see MyCustomHandler example)
        Handler handler = new MyCustomHandler("MyOutputFile.log");

        // Set up a custom Filter (see MyCustomFilter example)
        Vector acceptableLevels = new Vector();
        acceptableLevels.add(Level.INFO);
        acceptableLevels.add(Level.SEVERE);
        Filter filter = new MyCustomFilter(acceptableLevels);

        // Set up a custom Formatter (see MyCustomFormatter example)
        Formatter formatter = new MyCustomFormatter();

        // Connect the filter and formatter to the handler
        handler.setFilter(filter);
        handler.setFormatter(formatter);

        // Connect the handler to the logger
        logger.addHandler(handler);

        // avoid sending events logged to com.myCompany showing up in WebSphere
        // Application Server logs
        logger.setUseParentHandlers(false);
    }

    public static void main(String[] args) {
        initializeLogging();

        Logger logger = Logger.getLogger("com.myCompany");

        logger.info("This is a test INFO message");
        logger.warning("This is a test WARNING message");
        logger.logp(Level.SEVERE, "MyCustomLogging", "main", "This is a test SEVERE message");
    }
}

```

When the above program is run, the output of the program is written to the `MyOutputFile.log` file. The content of the log is in the expected log file, as controlled by the custom handler, and is formatted as defined by the custom formatter. The warning message is filtered out, as specified by the configuration of the custom filter. The output is as follows:

```
C:\>type MyOutputFile.log
Sat Sep 04 11:21:19 EDT 2004 INFO This is a test INFO message
Sat Sep 04 11:21:19 EDT 2004 SEVERE This is a test SEVERE message
```

Loggers

Loggers are used by applications and runtime components to capture message and trace events.

When situations occur that are significant either due to a change in state, for example when a server completes startup or because a potential problem is detected, such as a timeout waiting for a resource, a message is written to the logs. Trace events are logged in debugging scenarios, where a developer needs a clear view of what is occurring in each component to understand what might be going wrong. Logged events are often the only events available when a problem is first detected, and are used during both problem recovery and problem resolution.

Loggers are organized hierarchically. Each logger can have zero or more child loggers.

Loggers can be associated with a resource bundle. If specified, the resource bundle is used by the logger to localize messages that are logged to the logger. If the resource bundle is not specified, a logger uses the same resource bundle as its parent.

You can configure loggers with a level. If specified, the level is compared by the logger to incoming events. The events that are less severe than the level set for the logger are ignored by the logger. If the level is not specified, a logger takes on the level that is used by its parent. The default level for loggers is `Level.INFO`.

Loggers can have zero or more attached handlers. If supplied, all events that are logged to the logger are passed to the attached handlers. Handlers write events to output destinations such as log files or network sockets. When a logger finishes passing a logged event to all of the handlers that are attached to that logger, the logger passes the event to the handlers that are attached to the parents of the logger. This process stops if a parent logger is configured not to use its parent handlers. Handlers in WebSphere Application Server are attached to the root logger. Set the `useParentHandlers` logger property to `false` to prevent the logger from writing events to handlers that are higher in the hierarchy.

Loggers can have a filter. If supplied, the filter is invoked for each incoming event to tell the logger whether or not to ignore it.

Applications interact directly with loggers to log events. To obtain or create a logger, a call is made to the `Logger.getLogger` method with a name for the logger. Typically, the logger name is either the package qualified class name or the name of the package that the logger is used by. The hierarchical logger namespace is automatically created by using the dots in the logger name. For example, the `com.ibm.websphere.ras` logger has a `com.ibm.websphere` parent logger, which has a `com.ibm` parent. The parent at the top of the hierarchy is referred to as the *root logger*. This root logger is created during initialization. The root logger is the parent of the `com` logger.

Loggers are structured in a hierarchy. Every logger, except the root logger, has one parent. Each logger can also have 0 or more children. A logger inherits log handlers, resource bundle names, and event filtering settings from its parent in the hierarchy. The logger hierarchy is managed by the `LogManager` function.

Loggers create log records. A log record is the container object for the data of an event. This object is used by filters, handlers, and formatters in the logging infrastructure.

The logger provides several sets of methods for generating log messages. Some log methods take only a level and enough information to construct a message. Other, more complex logp (log precise) methods support the caller in passing class name and method name attributes, in addition to the level and message information. The logrb (log with resource bundle) methods add the capability of specifying a resource bundle as well as the level, message information, class name, and method name. Using methods such as severe, warning, fine, finer, and finest you can log a message at a particular level. For more information on logging and how to use it in your applications read “Using Java logging in an application” on page 34. For a complete list of methods, see the java.util.logging documentation at <http://java.sun.com/javase/>.

Log handlers

Log handlers write log record objects to output devices like log files, sockets, and notification mechanisms.

Loggers can have zero or more attached handlers. All objects that are logged to the logger are passed to the attached handlers, if handlers are supplied.

You can configure handlers with a level. The handler compares the level that is specified in the logged object to the level that is specified for the handler. If the level of the logged object is less severe than the level set in the handler, the object is ignored by the handler. The default level for handlers is ALL.

Handlers can have a filter. If a filter is supplied, the filter is invoked for each incoming object to tell the handler whether or not to ignore it.

Handlers can have a formatter. If a formatter is supplied, the formatter controls how the logged objects are formatted. For example, the formatter can decide to first include the time stamp, followed by a string representation of the level, followed by the message that is included in the logged object. The handler writes this formatted representation to the output device.

Both loggers and handlers can have levels and filters, and a logged object must pass all of these elements to be output. For example, you can set the logger level to FINE, but if the handler level is set at WARNING, only WARNING level messages are displayed in the output for that handler. Conversely, if your log handler is set to output all messages (level=All), but the logger level is set to WARNING, the logger never sends messages beneath the WARNING to the log handler.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using SystemOut.log , SystemErr.log, trace.log, and activity.log files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Log levels

Levels control which events are processed by Java logging. WebSphere Application Server controls the levels of all loggers in the system.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using SystemOut.log , SystemErr.log, trace.log, and activity.log files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The level value is set from configuration data when the logger is created and can be changed at run time from the administrative console. If a level is not set in the configuration data, a level is obtained by proceeding up the hierarchy until a parent with a level value is found. You can also set a level for each

handler to indicate which events are published to an output device. When you change the level for a logger in the administrative console, the change is propagated to the children of the logger.

Levels are cumulative; a logger can process logged objects at the level that is set for the logger, and at all levels above the set level.

Table 3. Valid log levels. This table lists valid logging levels.

Level	Content / Significance
Off	No events are logged.
Fatal	Task cannot continue and component cannot function.
Severe	Task cannot continue, but component can still function
Warning	Potential error or impending error
Audit	Significant event affecting server state or resources
Info	General information outlining overall task progress
Config	Configuration change or status
Detail	General information detailing subtask progress
Fine	Trace information - General trace
Finer	Trace information - Detailed trace + method entry / exit / return values
Finest	Trace information - A more detailed trace - Includes all the detail that is needed to debug problems
All	All events are logged. If you create custom levels, All includes your custom levels, and can provide a more detailed trace than Finest.

For instructions on how to set logging levels, read the topic about configuring Java logging using the administrative console.

Note: Trace information, which includes events at the Fine, Finer and Finest levels, can be written only to the trace log. Therefore, if you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest does not effect the logged data.

Log filters

Log filters help control more detailed logging settings that are not handled by usual log level settings.

A filter provides an optional, secondary control over what is logged, beyond the control that is provided by setting the level. Applications can apply a filter mechanism to control logging output through the logging APIs. An example of filter usage is to suppress all the events with a particular message key.

A filter is attached to a logger or log handler using the appropriate `setFilter` method. For a complete list of filter methods, see the `java.util.logging` documentation at <http://java.sun.com/javase/>

Log formatters

Log formatters format log messages so they can be used by various log handlers.

Handlers can be configured with a log formatter that knows how to format log records. The event, which is represented by the log record object, is passed to the appropriate formatter by the handler. The formatter returns formatted output to the handler, which writes the output to the output device.

The formatter is responsible for rendering the event for output. This formatter uses the resource bundle that is specified in the event to look up the message in the appropriate language.

Formatters are attached to handlers using the `setFormatter` method.

IBM i You can select a formatter for a handler using the administrative console panels. See Diagnostic trace service settings for details.

You can find the `java.util.logging` documentation at <http://java.sun.com/javase/>.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Java logging

Java logging is the logging toolkit that is provided by the `java.util.logging` package. Java logging provides a standard logging API for your applications.

Message logging (messages) and diagnostic trace (trace) are conceptually similar, but do have important differences. These differences are important for application developers to understand to use these tools properly. The following operational definitions of messages and trace are provided.

Message

A message entry is an informational record that is intended for end users, systems administrators, and support personnel to view. The text of the message must be clear, concise, and interpretable by an end user. Messages are typically localized and displayed in the national language of the end user. Although the destination and lifetime of messages might be configurable, enable some level of message logging in normal system operation. Use message logging judiciously because of performance considerations and the size of the message repository.

Trace A trace entry is an information record that is intended for service engineers or developers to use. As such, a trace record might be considerably more complex, verbose, and detailed than a message entry. Localization support is typically not used for trace entries. Trace entries can be fairly inscrutable, understandable only by the appropriate developer or service personnel. It is assumed that trace entries are not written during normal runtime operation, but can be enabled as needed to gather diagnostic information.

The application server redirects the system streams at the server startup. There is no way to allow the application to output logging to the console because the system streams can not be obtained by the application. If you would like to use console to monitor the application without using the console handler, you can either monitor the `SystemOut.log` file, or monitor a file created by another file handler.

Note: The application server uses Java logging internally and therefore certain restrictions apply for using system streams with this logging API by applications. During server startup, the standard output and error streams are replaced with special streams that write to the logging infrastructure, in order to include the output of the system streams in the log files. Because of this, applications can not use `java.util.logging.ConsoleHandler`, or any handler writing to `SystemErr.log` or `System.out` streams, attached to the root logger. If the user does attach the handler to the root logger, an infinite loop is created within the logging infrastructure, leading to stack overflow and server crash.

If the use of a handler that writes to system streams is necessary, attach it to a non-root logger so that it does not publish log records to parent handlers. The data written to the system streams is then formatted and written to the corresponding system stream log file. To monitor what is being written system streams, the configured log files (`SystemOut.log` and `SystemErr.log` by default) can be monitored.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace

infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Configuring the logger hierarchy

WebSphere Application Server handlers are attached to the Java root logger, which is at the top of the logger hierarchy. As a result, any request from anywhere in the logger tree can be processed by WebSphere Application Server handlers.

About this task

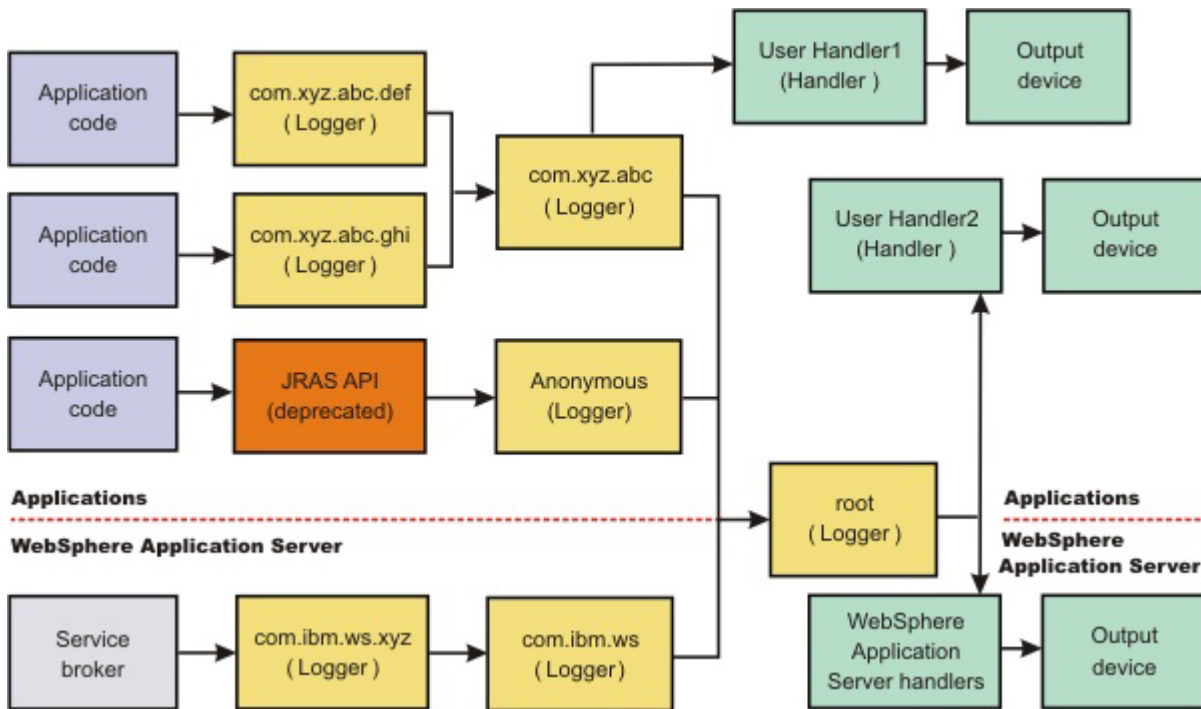
You can configure your application server to handle logs in many different ways. Configure your log settings based upon your configuration and the logging structure that best suits your needs.

Procedure

- Forward all application logging requests to the WebSphere Application Server handlers. This behavior is the default.
- Forward all application logging requests to your own custom handlers. Set the `useParentHandlers` option to `false` on one of your custom loggers, and then attach your handlers to that logger.
- Forward all application logging requests to both WebSphere Application Server handlers, and your custom handlers, but do not forward WebSphere Application Server logging requests to your custom handlers. Set the `useParentHandlers` option to `true` on one of your non-root custom loggers, and then attach your handlers to that logger. `True` is the default setting.
- Forward all WebSphere Application Server logging requests to both WebSphere Application Server handlers, and your custom handlers. Logging requests are always forwarded to WebSphere Application Server handlers. To forward WebSphere Application Server requests to your custom handlers, attach your custom handlers to the Java root logger, so that they are at the same level in the hierarchy as the WebSphere Application Server handlers.

Example

The following example shows how these requirements can be met using the Java logging infrastructure:



Creating log resource bundles and message files

You can forward messages that are written to the internal WebSphere Application Server logs to other processes for display. Messages that are displayed on the administrative console, which can be running in a different location than the server process, can be localized using the *late binding* process. Late binding means that WebSphere Application Server does not localize messages when they are logged, but defers localization to the process that displays the message.

About this task

Every method that accepts messages localizes those messages. The mechanism for providing localized messages is the resource bundle support provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with resource bundles as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the `java.util.ResourceBundle`, `java.util.ListResourceBundle` and `java.util.PropertyResourceBundle` classes, as well as the `java.text.MessageFormat` class.

The `PropertyResourceBundle` class is the preferred mechanism to use.

To properly localize the message, the displaying process must have access to the resource bundle where the message text is stored. You must package the resource bundle separately from the application, and install it in a location where the viewing process can access it.

By default, the WebSphere Application Server runtime localizes all the messages when they are logged. This localization eliminates the need to pass a `.jar` file to the application, unless you need to localize in a different location. However, you can use the early binding technique to localize messages as they log. An application that uses early binding must localize the message before logging it. The application looks up

the localized text in the resource bundle and formats the message. Use the early binding technique to package the application resource bundles with the application.

To create a resource bundle, perform the following steps.

Procedure

1. Create a text properties file that lists message keys and the corresponding messages. The properties file must have the following characteristics:
 - Each property in the file is terminated with a line-termination character.
 - If a line contains white space only, or if the first non-white space character of the line is the pound sign symbol (#) or exclamation mark (!), the line is ignored. The # and ! characters can therefore be used to put comments into the file.
 - Each line in the file, unless it is a comment or consists of white space only, denotes a single property. A backslash (\) is treated as the line-continuation character.
 - The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (=), colon (:), and white space ().
 - The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (\), but doing this process is not recommended, because escaping characters is error prone and confusing. Instead, use a valid separator character that does not display in any keys in the properties file.
 - White space after the key and separator is ignored until the first non-white space character is encountered. All characters remaining before the line-termination character define the element.

See the Java documentation for the `java.util.Properties` class for a full description of the syntax and the construction of properties files.

2. Translate the file into localized versions of the file with language-specific file names. For example, a file named `DefaultMessages.properties` can be translated into `DefaultMessages_de.properties` for German and `DefaultMessages_ja.properties` for Japanese.
3. When the translated resource bundles are available, put the bundle in a directory that is part of the application class path.
4. When a message logger is obtained from the log manager, configure it to use a particular resource bundle. Messages logged with the Logger API use this resource bundle when message localization is performed. At run time, the user locale setting determines the properties file from which to extract the message that is specified by a message key, ensuring that the message is delivered in the correct language.
5. If the message loggers `msg` method is called, a resource bundle name must be explicitly provided.

Example

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a properties resource bundle. This example shows how to create such a properties file.

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs are inserted. All the normal properties file conventions and rules apply to this file. In addition, the creator must be aware of other restrictions that are imposed on the values by the Java `MessageFormat` class. For example, apostrophes must be escaped or they cause a problem. Avoid the use of non-portable characters. WebSphere Application Server does not support the use of extended formatting conventions that the `MessageFormat` class supports, such as `{1, date}` or `{0,number, integer}`.

Assume that the base directory for the application that uses this resource bundle is `baseDir` and that this directory is in the class path. Assume that the properties file is stored in the subdirectory `baseDir` that is not in the class path (for example, `baseDir/subDir1/subDir2/resources`). To allow the messages file to resolve, the `subDir1.subDir2.resources.DefaultMessage` name is used to identify the property resource bundle and is passed to the message logger.

For this sample, the properties file is named `DefaultMessages.properties`.

```
# Contents of the DefaultMessages.properties file
MSG_KEY_00=A message with no substitution parameters.
MSG_KEY_01=A message with one substitution parameter: parm1={0}
MSG_KEY_02=A message with two substitution parameters: parm1={0}, parm2 = {1}
MSG_KEY_03=A message with three parameter: parm1={0}, parm2 = {1}, parm3={2}
```

When the `DefaultMessages.properties` file is created, the file can be sent to a translation center where the localized versions are generated.

What to do next

The application locates the resource bundle based on the file location relative to any directory in the class path. For instance, if the `DefaultMessages.properties` property resource bundle is located in the `baseDir/subDir1/subDir2/resources` directory and `baseDir` is in the class path, the name `subdir1.subdir2.resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

Logger.properties file for configuring logger settings

Use the `Logger.properties` file to set logger attributes for specific loggers.

The properties file is loaded the first time that the `Logger.getLogger(logger_name)` method is called within an application.

Important: The name of the `Logger.properties` file is case sensitive. Use a capital "L" in the file name.

When an application calls the `Logger.getLogger` method for the first time, all the available logger properties files are loaded. Applications can provide `Logger.properties` files in:

- the META-INF directory of the Java archive (JAR) file for the application
- directories included in the class path of an application module
- directories included in the application class path

The properties file contains two categories of parameters, logger control and logger data:

- Logger control information
 - Minimum localization level: The minimum `LogRecord` level for which localization is attempted
 - Group: The logical group that this component belongs to
 - Event factory: The Common Base Event template file to use with the event factory. The naming convention for this template is the fully qualified component name, with a file extension of `.event.xml`. For example, a template that applies to the `com.ibm.compXYZ` package is called `com.ibm.compXYZ.event.xml`.
- Logger data information
 - Product name
 - Organization name
 - Component name
 - Extensions and additional properties

Syntax of the Logger.properties file

Use the following syntax to set logger properties:

```
<logger base name>.<property>=value
```

where:

logger base name is the starting part of the logger name to which the property applies. All loggers with names starting with this string have the property applied.

property is one of the following properties:

- organization
- product
- component
- minimum_localization_level
- group
- eventfactory

Sample Logger.properties file

In the following sample, the `com.ibm.xyz.MyEventFactory` event factory is used by any loggers in the `com.ibm.websphere.abc` package or any sub packages that do not override this value in their configuration file.

```
com.ibm.websphere.abc.eventfactory=com.ibm.xyz.MyEventFactory
```

Group Logger.properties file

In the following example, the group is `MyTraceGroup` and the components are `com.ibm.stuff` and `com.ibm.morestuff`:

```
com.ibm.stuff.group=MyTraceGroup
com.ibm.morestuff.group=MyTraceGroup
```

Configuring applications to use Jakarta Commons Logging

Jakarta Commons Logging provides a simple logging interface and thin wrappers for several logging systems. WebSphere Application Server supports Jakarta Commons Logging by providing a logger. The support does not change interfaces defined by Jakarta Commons Logging.

Before you begin

The WebSphere Application Server logger is a thin wrapper for the WebSphere Application Server logging facility. The logger name is `com.ibm.websphere.commons.logging.WsJDK14Logger`. The logger can handle logging objects defined by either of the following:

- Java Logging found in Java Specification Request 47: Logging API Specification
- Common Base Event

A *logging object* is an object that holds logging entry information.

To better understand Jakarta Commons Logging, read Jakarta Commons and the specifications for Java Logging and for Common Base Event. To better understand use of the WebSphere Application Server logger, read “Jakarta Commons Logging” on page 50.

About this task

WebSphere Application Server provides the Jakarta Commons Logging binary distribution in its `libraries` directory. By default, the product uses the Jakarta Commons Logging `LogFactory` implementation and `JDK14Logger`.

best-practices: The default configuration of Jakarta Commons Logging is stored in the `commons-logging.properties` file. To specify the factory class to use with Jakarta Commons Logging in an application, provide a file named

`org.apache.commons.logging.LogFactory`, located in `META-INF/services` directory, that contains the name of the factory class on the first line. This is the configuration mechanism for the JAR file service provider, as defined in JDK 1.3 and above.

For an application to use the WebSphere Application Server logger, the application must provide its own configuration for the logger. To configure an application to use the WebSphere Application Server logger, complete the steps that follow.

Procedure

1. Examine “Configurations for the WebSphere Application Server logger” on page 53 and determine which configuration best suits your application.
2. Change your application configuration as needed to enable use of the WebSphere Application Server logger.

Results

After the application starts, Jakarta Commons Logging routes the application's logging output to the WebSphere Application Server logger.

Jakarta Commons Logging

Jakarta Commons Logging provides a simple logging interface and thin wrappers for several logging systems. The logging interface enables application logging to be simple and independent of the logging system that the application uses. You can change the logging implementation for a deployed application without having to change the application logging code. However, the simplicity of the logging interface prevents the application from leveraging all the functionality of the logging systems.

This topic provides the following information about Jakarta Commons Logging in WebSphere Application Server:

- “Support for Jakarta Commons Logging”
- “Benefits of support for Jakarta Commons Logging”
- “Overview of the process for using Jakarta Commons Logging” on page 51
- “Classes used to obtain a logger factory and logger” on page 51
- “Logger level configuration and mapping” on page 52

Support for Jakarta Commons Logging

The product supports Jakarta Commons Logging by providing a logger, a thin wrapper for the WebSphere Application Server logging facility. The logger can handle both Java Logging (JSR-47) and Common Base Event logging objects. A *logging object* is an object that holds logging entry information.

The product support for Jakarta Commons Logging does not change interfaces defined by Jakarta Commons Logging.

Benefits of support for Jakarta Commons Logging

The WebSphere Application Server support for Jakarta Commons Logging provides the following benefits:

- WebSphere Application Server is pre-configured to use Jakarta Commons Logging.
All of the functionality of Jakarta Commons Logging is provided for any application or WebSphere Application Server component. Logging calls are routed by default to the underlying WebSphere Application Server logging facility.
- A logger that uses the WebSphere Application Server logging facility.

Applications and components can pass both Java Logging and Common Base Event logging objects to the WebSphere Application Server logger without conversion to strings, providing applications with enhanced logging. Further, Jakarta Commons Logging Logger levels are integrated into WebSphere Application Server administrative facilities.

Overview of the process for using Jakarta Commons Logging

Logging with Jakarta Commons Logging consists of the steps that follow. “Configurations for the WebSphere Application Server logger” on page 53 provides details on configuring your application to use the WebSphere Application Server logger.

1. Obtain an instance of a logger factory.

To obtain a logger factory, use Jakarta Commons Logging code. You can configure the code to meet your needs. In WebSphere Application Server, Jakarta Commons Logging is configured by default to instantiate the Jakarta Commons Logging default logger factory. Applications or WebSphere Application Server components can provide their own configuration if they use a different logger factory implementation. Applications can use more than one factory.

2. Obtain an instance of a logger.

To obtain a logger, use code implemented by a logger factory. Configuration of the code is implementation specific.

The WebSphere Application Server logger implements the methods defined in the logging interface. The logging methods take at least one argument, which can be any Java object. The WebSphere Application Server logger, the `WsJDK14Logger` logger described in “Classes used to obtain a logger factory and logger,” handles the following objects passed into the following logging methods:

CommonBaseEvent

Wrapped into `CommonBaseEventLogRecord`

CommonBaseEventLogRecord

Passed without change

LogRecord

Passed without change

Other objects

Converted to `String`

Applications or WebSphere Application Server components can provide their own configuration if they use an implementation of a logger that is not specific to WebSphere Application Server. An application must know what factory is being used in order to configure it.

3. Start your application. Jakarta Commons Logging routes the application's logging output to the designated logger

Classes used to obtain a logger factory and logger

Table 4. Jakarta Commons Logging class descriptions. Use the classes for a logger factory instance and logger.

Class name	Description
LogFactory	<p><i>LogFactory</i> is a Jakarta Commons Logging class that implements initialization logic. <i>LogFactory</i> is an abstract class that every logger factory implementation has to extend. It provides static methods for obtaining:</p> <ul style="list-style-type: none"> • An instance of a factory class • Instances of a logger, using an instance of the factory class <p><i>LogFactory</i> provides methods for obtaining instances of loggers, although these methods delegate the logger instantiation and configuration to an instance of a logger factory class.</p> <p>Logger factories, once instantiated, are cached on a per context class loader basis. The instances in a cache can be released. This functionality is designed for platform container implementations rather than for applications.</p>

Table 4. Jakarta Commons Logging class descriptions (continued). Use the classes for a logger factory instance and logger.

Class name	Description
LogFactoryImpl	<i>LogFactoryImpl</i> is a Jakarta Commons Logging concrete class that implements the default logger factory using methods in LogFactory. To use Java Logging, there must always be at least one instance of a logger factory class, even if the application has not explicitly obtained one. If the configuration does not name a logger factory class, LogFactoryImpl is used as the default.
Log	<p><i>Log</i> is a Jakarta Commons Logging interface for loggers. Commons logging loggers have to implement the Log interface. Because the goal of Jakarta Commons Logging is to wrapper any logging system, the Log interface defines a small set of common logging methods. In WebSphere Application Server, WsJDK14Logger implements the Log interface.</p> <p>Logger instantiation and configuration is specific to every logger factory. Logging in WebSphere Application Server uses the default logger factory provided in Jakarta Commons Logging, which keeps instantiated loggers in cache, on a per context class loader basis.</p>
WsJDK14Logger	<i>WsJDK14Logger</i> is a WebSphere Application Server class that provides a Jakarta Commons Logging logger by implementing the Log interface. The WsJDK14Logger logger differs from the Java Logging logger in that the WsJDK14Logger logger enables Java Logging or Common Base Event objects to be passed over without converting them into String objects. This prevents any information loss the conversion to String might cause as well as allows the logging output to be more descriptive and precise. In contrast, the Java Logginglogger that is provided in Jakarta Commons Logging converts objects passed into the logging calls to String objects prior to passing them over to the underlying Java Logging.

Logger level configuration and mapping

Because Jakarta Commons Logging loggers are thin wrappers for specific logging systems, the loggers do not have their own level, but use the level of the logger from the underlying logging system. Although the underlying system can provide methods for changing level, there are no methods for changing level defined on the Log interface, which all Jakarta Commons Logging loggers must implement. WsJDK14Logger uses the level of its underlying Java Logging logger.

Following table shows, on the left, the mapping of Jakarta Commons Logging levels within WsJDK14Logger to levels in the WebSphere Application Server implementation of Java Logging. The first column shows the levels defined in Java Logging and the level mapping in the Jakarta Commons Logging JDK14Logger to the Java Logging levels.

Table 5. Mapping of WsJDK14Logger levels to Java Logging levels. Compare the logging levels.

WsJDK14Logger	Java Logging in WebSphere Application Server	Java Logging	JDK14Logger
Fatal	Fatal		
Error	Severe	Severe	Fatal, Error
Warning	Warning	Warning	Warning
	Audit		
Info	Info	Info	Info
	Config	Config	
	Detail		
Debug	Fine	Fine	Debug
	Finer	Finer	
Trace	Finest	Finest	Trace

The WsJDK14Logger level is synchronized with the underlying Java Logging logger level. WebSphere Application Server administration controls the WsJDK14Logger level.

Configurations for the WebSphere Application Server logger

This topic describes several ways to configure an application to use the WebSphere Application Server logger.

The type of configuration that best suits an application depends upon the following:

- Whether the class loader order setting for the application is `Classes loaded with parent class loader first (Parent First)` or `Classes loaded with application class loader first (Parent Last)`, you can set the class loader delegation mode on a console page. For more details about class load order and delegation, consult the class loading chapter in the *Developing applications* PDF book.
- Whether Jakarta Commons Logging is bundled with the application configuration
- Whether Jakarta Commons Logging is provided within the application

The following tables describe the conditions required to enable an application to use the WebSphere Application Server logger.

Class loader mode is Parent First and Jakarta Commons Logging is bundled with the application

Table 6. Conditions required to use logger. When Parent First and Jakarta Commons Logging is bundled with an application.

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
<p>The application provides the configuration by either of the following:</p> <ul style="list-style-type: none"> • The properties file <code>commons-logging.properties</code> in the application classpath is not read by the LogFactory because the parent class loader finds the WebSphere properties file first. • The class name is read from the file <code>META-INF/services/org.apache.commons.logging.LogFactory</code> 	<p>The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration, unless the configuration is provided in a META-INF file of the application or module.</p>	<p>The log used is either of the following:</p> <ul style="list-style-type: none"> • The Log implementation specified in the WebSphere Application Server default configuration • An application-specific Log implementation if an application-specific LogFactory that instantiates a different Log implementation is used. 	<p>The application parent class loader is the first class loader to load the Jakarta Commons Logging code. The WebSphere bundle that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes.</p> <p>For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the parent class loader.</p>
<p>Not provided by the application</p>	<p>The log factory used is the LogFactory implementation specified in the WebSphere default configuration.</p>	<p>The log used is the Log implementation specified in the WebSphere default configuration.</p>	<p>The Jakarta Commons Logging bundled with the application is not used.</p>

Class loader mode is Parent First and Jakarta Commons Logging is not bundled with the application

Table 7. Conditions required to use logger. When Parent First and Jakarta Commons Logging is not bundled with an application.

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
<p>The application provides the configuration by either of the following:</p> <ul style="list-style-type: none"> The properties file <code>commons-logging.properties</code> in the application classpath is not read by the LogFactory because the parent class loader finds the WebSphere Application Server properties file first. The class name is read from the file <code>META-INF/services/org.apache.commons.logging.LogFactory</code> 	<p>The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration, unless the configuration is provided in a META-INF file of the application or module.</p>	<p>The log used is either of the following:</p> <ul style="list-style-type: none"> The Log implementation specified in the WebSphere Application Server default configuration An application-specific Log implementation if an application-specific LogFactory that instantiates a different Log implementation is used. 	<p>The application parent class loader is the first class loader to load the Jakarta Commons Logging code. The WebSphere bundle that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes.</p> <p>For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the parent class loader.</p>
<p>Not provided by the application</p>	<p>The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.</p>	<p>The log used is the Log implementation specified in the WebSphere Application Server default configuration.</p>	<p>Same as in the previous row</p>

Class loader mode is Parent Last and Jakarta Commons Logging is bundled with the application

Table 8. Conditions required to use logger. When Parent Last and Jakarta Commons Logging is bundled with an application.

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
<p>The application provides the configuration by either of the following:</p> <ul style="list-style-type: none"> The properties file <code>commons-logging.properties</code> in the application classpath is read by the LogFactory because the class loader finds the application properties file first. The class name is read from the file <code>META-INF/services/org.apache.commons.logging.LogFactory</code> 	<p>The log factory used is either of the following:</p> <ul style="list-style-type: none"> The default Jakarta Commons Logging LogFactory The LogFactory specified in the application configuration 	<p>The log used is the Log implementation specified in the application configuration.</p> <p>If the log factory used is the default Jakarta Commons Logging LogFactory, the Log implementation must be on the classpath of the application class loader.</p>	<p>The application class loader is the first class loader to load the Jakarta Commons Logging code. The application bundle that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes.</p> <p>For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the application class loader.</p>
<p>Not provided by the application</p>	<p>The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.</p>	<p>The log used is the Log implementation specified in the WebSphere Application Server default configuration.</p>	

Class loader mode is Parent Last and Jakarta Commons Logging is not bundled with the application

Table 9. Conditions required to use logger. When Parent Last and Jakarta Commons Logging is not bundled with an application.

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
<p>The application provides the configuration by either of the following:</p> <ul style="list-style-type: none"> The properties file <code>commons-logging.properties</code> in the application classpath is read by the LogFactory because the class loader finds the application properties file first. The class name is read from the file <code>META-INF/services/org.apache.commons.logging.LogFactory</code> 	<p>The log factory used is either of the following:</p> <ul style="list-style-type: none"> The default Jakarta Commons Logging LogFactory The LogFactory specified in the application configuration 	<p>The log used is the Log implementation specified in the application configuration.</p> <p>If the log factory used is the default Jakarta Commons Logging LogFactory, the Log implementation must be on the classpath of the application class loader.</p>	<p>There is no Jakarta Commons Logging code at the application class loader. Thus, the WebSphere bundle that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes.</p> <p>For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the parent class loader.</p>
Not provided by the application	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.	The log used is the Log implementation specified in the WebSphere Application Server default configuration.	

Programming with the JRas framework

Use the JRas extensions to incorporate message logging and diagnostic trace into WebSphere Application Server applications.

Before you begin

Note: The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

The JRas extensions allow message logging and diagnostic trace to work with WebSphere Application Server applications. They are based on the stand-alone JRas logging toolkit.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Retrieve a reference to the JRas manager.
2. Retrieve message and trace loggers by using methods on the returned manager.
3. Call the appropriate methods on the returned message and trace loggers to create message and trace entries, as appropriate.

JRas logging toolkit

The JRas logging toolkit provides diagnostic information to help the administrator diagnose problems or tune application performance.

Note: The JRes framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

Developing, deploying, and maintaining applications are complex tasks. For example, when a running application encounters an unexpected condition, it might not be able to complete a requested operation. In such a case, you might want the application to inform the administrator that the operation failed and provide information. This action enables the administrator to take the proper corrective action. Those who develop or maintain applications might need to gather detailed information relating to the path of a running application to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as *message logging* and *diagnostic trace*.

Message logging (messages) and diagnostic trace (trace) are conceptually quite similar, but do have important differences. It is important for application developers to understand these differences to use these tools properly. To start with, the following operational definitions of messages and trace are provided.

Message

A message entry is an informational record that is intended for end users, systems administrators and support personnel to view. The text of the message must be clear, concise, and interpretable. Messages are typically localized, meaning that they display in the national language of the end user. Although the destination and lifetime of messages might be configurable, some level of message logging is always enabled in normal system operation. Message logging must be used judiciously due to both performance considerations and the size of the message repository.

Trace A trace entry is an information record that is intended for service engineers or developers to use. This trace record might be considerably more complex, verbose, and detailed than a message entry. Localization support is typically not used for trace entries. Trace entries can be fairly inscrutable, understandable only by the appropriate developer or service personnel. It is assumed that trace entries are not written during normal runtime operation, but might be enabled as needed to gather diagnostic information.

WebSphere Application Server provides a message logging and diagnostic trace API that applications can use. This API is based on the stand-alone JRes logging toolkit, which was developed by IBM. The stand-alone JRes logging toolkit is a collection of interfaces and classes that provide message logging and diagnostic trace primitives. These primitives are not tied to any particular product or platform. The stand-alone JRes logging toolkit provides a limited amount of support, which is typically referred to as *systems management support*, including log file configuration support based on property files.

As designed, the stand-alone JRes logging toolkit does not contain the support that is required for integration into the WebSphere Application Server run time or for use in a Java 2 Platform, Enterprise Edition (J2EE) environment. To overcome these limitations, WebSphere Application Server provides a set of extension classes to address these shortcomings. This collection of extension classes is referred to as the JRes extensions. The JRes extensions do not modify the interfaces that are introduced by the stand-alone JRes logging toolkit, but provide the appropriate implementation classes. The conceptual structure that is introduced by the stand-alone JRes logging toolkit is described in the following section. It is equally applicable to the JRes extensions.

JRes concepts

The section contains a basic overview of important concepts and constructs that are introduced by the stand-alone JRes logging toolkit. This information is not an exhaustive overview of the capabilities of this logging toolkit, nor is it intended as a detailed discussion of usage or programming paradigms. More detailed information, including code examples, is available in JRes extensions and its subtopics, including in the API documentation for the various interfaces and classes that make up the logging toolkit.

Event types

The stand-alone JRes logging toolkit defines a set of event types for messages and a set of event types for trace. Examples of message types include informational, warning, and error. Examples of trace types include entry, exit, and trace.

Event classes

The stand-alone JRas logging toolkit defines both message and trace event classes.

Loggers

A logger is the primary object with which the user code interacts. Two types of loggers are defined: message loggers and trace loggers. The set of methods on message loggers and trace loggers are different because they provide different functionality. Message loggers create message records only and trace loggers create trace records only. Both types of loggers contain masks that indicate which categories of events the logger processes and which to ignore. Although every JRas logger is defined to contain both a message and trace mask, the message logger uses only the message mask and the trace logger uses the trace mask only. For example, by setting a message logger message mask to the appropriate state, it can be configured to process only error messages and ignore informational and warning messages. Changing the trace mask state of a message logger has no effect.

A logger contains one or more handlers to which it forwards events for further processing. When the user calls a method on the logger, the logger compares the event type that is specified by the caller to its current mask value. If the specified type passes the mask check, the logger creates an event object to capture the information relating to the event that passed to the logger method. This information can include information, such as the names of the class and method which logs the event, a message, and parameters to log, among others. When the logger creates the event object, it forwards the event to all handlers currently registered with the logger.

Methods that are used within the logging infrastructure do not make calls to the logger method. When an application uses an object that extends a thread class, implements the hashCode method, and makes a call to the logging infrastructure from that method, the result is a recursive loop.

Handlers

A handler provides an abstraction over an output device or event consumer. An example is a file handler, which knows how to write an event to a file. The handler also contains a mask that is used to further restrict the categories of events the handler processes. For example, a message logger might be configured to pass both warning and error events, but a handler attached to the message logger might be configured to pass error events only. Handlers also include formatters, which the handler invokes to format the data in the passed event before it is written to the output device.

Formatters

Handlers are configured with formatters, which know how to format events of certain types. A handler can contain multiple formatters, each of which knows how to format a specific class of event. The event object is passed to the appropriate formatter by the handler. The formatter returns formatted output to the handler, which then writes it to the output device.

JRas Extensions

JRas extensions are the collection of implementation classes that support JRas integration into the WebSphere Application Server environment.

JRas extensions

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The stand-alone JRas logging toolkit defines interfaces and provides a variety of concrete classes that implement these interfaces. Because the stand-alone JRas logging toolkit is developed as a general purpose toolkit, the implementation classes do not contain the configuration interfaces and methods that are necessary for use in the WebSphere Application Server product. In addition, many of the implementation classes are not written appropriately for use in a Java 2 Platform, Enterprise Edition (J2EE) environment. To overcome these shortcomings, WebSphere Application Server provides the appropriate implementation classes that support integration into the WebSphere Application Server

environment. The collection of these implementation classes is referred to as the *JRas extensions*.

Usage model

You can use the JRas extensions in three distinct operational modes:

Integrated

In this mode, message and trace records are written only to logs that are defined and maintained by the WebSphere Application Server run time. This mode is the default mode of operation and is equivalent to the WebSphere Application Server V4.0 mode of operation.

Stand-alone

In this mode, message and trace records are written solely to stand-alone logs that are defined and maintained by the user. You control which categories of events are written to which logs, and the format in which entries are written. You are responsible for configuration and maintenance of the logs. Message and trace entries are not written to WebSphere Application Server runtime logs.

Combined

In this mode, message and trace records are written to both WebSphere Application Server runtime logs and to stand-alone logs that you must define, control, and maintain. You can use filtering controls to determine which categories of messages and trace are written to which logs.

The JRas extensions are specifically targeted to an integrated mode of operation. The integrated mode of operation can be appropriate for some usage scenarios, but many scenarios are not adequately addressed by these extensions. Many usage scenarios require a stand-alone or combined mode of operation instead. A set of user extension points are defined that support JRas extensions in either a stand-alone or combined mode of operations.

JRas extension classes

WebSphere Application Server provides a base set of implementation classes that are collectively referred to as the *JRas extensions*. Many of these classes provide the appropriate implementations of loggers, handlers, and formatters for use in a WebSphere Application Server environment.

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The collection of JRas classes is targeted at an integrated mode of operation. If you choose to use the JRas extensions in either stand-alone or combined mode, you can reuse the logger and manager class that are provided by the extensions, but you must provide your own implementations of handlers and formatters.

WebSphere Application Server message and trace loggers

The message and trace loggers that are provided by the stand-alone JRas logging toolkit cannot be directly used in the WebSphere Application Server environment. The JRas extensions provide the appropriate logger implementation classes. Instances of these message and trace logger classes are obtained directly and exclusively from the WebSphere Application Server Manager class. You cannot directly instantiate message and trace loggers. Obtaining loggers in any manner other than directly from the Manager class is not allowed and directly violates the programming model.

The message and trace logger instances that are obtained from the WebSphere Application Server Manager class are subclasses of the `RASMessageLogger` and `RASTraceLogger` classes that are provided by the stand-alone JRas logging toolkit. The `RASMessageLogger` and `RASTraceLogger` classes define the set of methods that are directly available. Public methods that are introduced by the JRas extensions logger subclasses cannot be called directly by user code because it is a violation of the programming model.

Loggers are named objects and are identified by name. When the Manager class is called to obtain a logger, the caller is required to specify a name for the logger. The Manager class maintains a

name-to-logger instance mapping. Only one instance of a named logger is ever created within the lifetime of a process. The first call to the Manager class with a particular name results in the logger, which is configured by the Manager class. The Manager class caches a reference to the instance, then returns it to the caller. Subsequent calls to the Manager class that specify the same name result in a returned reference to the cached logger. Separate namespaces are maintained for message and trace loggers. You can use a single name obtain both a message logger and a trace logger from the Manager, without ambiguity, and without causing a namespace collision.

In general, loggers have no predefined granularity or scope. A single logger can be used to instrument an entire application. You might determine that having a logger per class is more effective, or the appropriate granularity might be somewhere in between. Partitioning an application into logging domains is determined by the application writer.

The WebSphere Application Server logger classes that are obtained from the Manager class are thread-safe. Although the loggers provided as part of the stand-alone JRas logging toolkit implement the serializable interface, loggers are not serializable. Loggers are stateful objects, tied to a Java virtual machine instance and are not serializable. Attempting to serialize a logger is a violation of the programming model.

Personal or individual logger subclasses are not supported in a WebSphere Application Server environment.

WebSphere Application Server handlers

WebSphere Application Server provides the appropriate handler class that is used to write message and trace events to the WebSphere Application Server run time logs. You cannot configure the WebSphere Application Server handler to write to any other destination. The creation of a WebSphere Application Server handler is a restricted operation and is not available to user code. Every logger that is obtained from the Manager comes preconfigured with an instance of this handler already installed. You can remove the WebSphere Application Server handler from a logger when you want to run in stand-alone mode. When you remove it, you cannot add the WebSphere Application Server handler again to the logger from which it is removed or any other logger. Also, you cannot directly call any method on the WebSphere Application Server handler. Attempting to create an instance of the WebSphere Application Server handler, to call methods on the WebSphere Application Server handler or to add a WebSphere Application Server handler to a logger by user code is a violation of the programming model.

WebSphere Application Server formatters

The WebSphere Application Server handler comes preconfigured with the appropriate formatter for data that is written to WebSphere Application Server logs. The creation of a WebSphere Application Server formatter is a restricted operation and not available to user code. No mechanism exists that allows the user to obtain a reference to a formatter installed in a WebSphere Application Server handler, or to change the formatter a WebSphere Application Server handler is configured to use.

WebSphere Application Server manager

WebSphere Application Server provides a Manager class in the `com.ibm.websphere.ras` package. All message and trace loggers must be obtained from this Manager class. A reference to the Manager class is obtained by calling the static `Manager.getManager` method. Message loggers are obtained by calling the `createRASMessageLogger` method on the Manager class. Trace loggers are obtained by calling the `createRASTraceLogger` method on the Manager class.

The manager also supports a *group* abstraction that is useful when dealing with trace loggers. The group abstraction supports multiple, unrelated trace loggers to register as part of a named entity called a *group*. WebSphere Application Server provides the appropriate systems management facilities to manipulate the trace setting of a group, similar to the way the trace settings of an individual trace logger work.

For example, suppose component A consists of 10 classes. Suppose each class is configured to use a separate trace logger. All 10 trace loggers in the component are registered as members of the same group, for example, `Component_A_Group`. You can turn on trace for a single class, or you can turn on trace for all 10 classes in a single operation using the group name, if you want a component trace. Group names are maintained within the namespace for trace loggers.

JRas framework (deprecated)

Because the JRas extensions classes do not provide the flexibility and behavior that are required for many scenarios, a variety of extension points are defined. You can write your own implementation classes to obtain the required behavior.

Deprecated: The JRas framework described in this topic is deprecated. However, you can achieve similar results using Java logging.

In general, the JRas extensions require you to call the Manager class to obtain a message logger or trace logger. No provision is made for you to provide your own message or trace logger subclasses. In general, user-provided extensions cannot be used to affect the integrated mode of operation. The behavior of the integrated mode of operation is solely determined by the WebSphere Application Server run time and the JRas extensions classes.

Handlers

The stand-alone JRas logging toolkit defines the `RASHandler` interface. All handlers must implement this interface. You can write your own handler classes that implement the `RASHandler` interface. Directly create instances of user-defined handlers and add them to the loggers that are obtained from the Manager class.

The stand-alone JRas logging toolkit provides several handler implementation classes. These handler classes are inappropriate for use in the Java 2 Platform, Enterprise Edition (J2EE) environment. You cannot directly use or subclass any of the Handler classes that are provided by the stand-alone JRas logging toolkit. Doing so is a violation of the programming model.

Formatters

The stand-alone JRas logging toolkit defines the `RASFormatter` interface. All formatters must implement this interface. You can write your own formatter classes that implement the `RASFormatter` interface. You can add these classes to a user-defined handler only. WebSphere Application Server handlers cannot be configured to use user-defined formatters. Instead, directly create instances of your formatters and add them to the your handlers appropriately.

As with handlers, the stand-alone JRas logging toolkit provides several formatter implementation classes. Direct use of these formatter classes is not supported.

Message event types

The stand-alone JRas toolkit defines message event types in the `RASMessageEvent` interface. In addition, the WebSphere Application Server reserves a range of message event types for future use. The `RASMessageEvent` interface defines three types, with values of `0x01`, `0x02`, and `0x04`. The values `0x08` through `0x8000` are reserved for future use. You can provide your own message event types by extending this interface appropriately. User-defined message types must have a value of `0x1000` or greater.

Message loggers that are retrieved from the Manager class have their message masks set to pass or process all message event types defined in the `RASMessageEvent` interface. To process user-defined message types, you must manually set the message logger mask to the appropriate state by user code after the message logger is obtained from the Manager class. WebSphere Application Server does not provide any built-in systems management support for managing message types.

Message event objects

The stand-alone JRas toolkit provides a `RASMessageEvent` implementation class. When a message logging method is called on the message logger, and the message type is currently enabled, the logger creates and distributes an event of this class to all handlers that are currently registered with that logger.

You can provide your own message event classes, but they must implement the `RASIEvent` interface. You must directly create instances of such user-defined message event classes. When it is created, pass your message event to the message logger by calling the message logger's `fireRASEvent` method directly. WebSphere Application Server message loggers cannot directly create instances of user-defined types in response to calling a logging method (`msg.message`) on the logger. In addition, instances of user-defined message types are never processed by the WebSphere Application Server handler. You cannot create instances of the `RASMessageEvent` class directly.

Trace event types

The stand-alone JRas toolkit defines trace event types in the `RASITraceEvent` interface. You can provide your own trace event types by extending this interface appropriately. In such a case, you must ensure that the values for the user-defined trace event types do not collide with the values of the types that are defined in the `RASITraceEvent` interface.

Trace loggers that are retrieved from the Manager class typically have their trace masks set to reject all types. A different starting state can be specified by using WebSphere Application Server systems management facilities. In addition, you can change the state of the trace mask for a logger at run-time, using WebSphere Application Server systems management facilities.

To process user-defined trace types, the trace logger mask must be manually set to the appropriate state by user code. WebSphere Application Server systems management facilities cannot be used to manage user-defined trace types, either at start time or run time.

Trace event objects

The stand-alone JRas toolkit provides a `RASTraceEvent` implementation class. When a trace logging method is called on the WebSphere Application Server trace logger and the type is currently enabled, the logger creates and distributes an event of this class to all the handlers that are currently registered with that logger.

You can provide your own trace event classes. Such trace event classes must implement the `RASIEvent` interface. You must create instances of such user-defined event classes directly. When it is created, pass the trace event to the trace logger by calling the trace logger's `fireRASEvent` method directly. WebSphere Application Server trace loggers cannot directly create instances of user-defined types in response to calling a trace method (`entry`, `exit`, `trace`) on the trace logger. In addition, instances of user-defined trace types are never processed by the WebSphere Application Server handler. You cannot create instances of the `RASTraceEvent` class directly.

User defined types, user defined events and WebSphere Application Server

By definition, the WebSphere Application Server handler processed user-defined message or trace types, or user-defined message or trace event classes. Message and trace entries of either a user-defined type or user-defined event class cannot be written to the WebSphere Application Server run-time logs.

JRas programming interfaces for logging (deprecated):

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

General considerations

You can configure the WebSphere Application Server to use Java 2 security to restrict access to protected resources such as the file system and sockets. Because user-written extensions typically access such protected resources, user-written extensions must contain the appropriate security checking calls, using `AccessController.doPrivileged` calls. In addition, the user-written extensions must contain the appropriate policy file. In general, locating user-written extensions in a separate package is a good practice. It is your responsibility to restrict access to the user-written extensions appropriately.

Writing a handler

User-written handlers must implement the `RASHandler` interface. The `RASHandler` interface extends the `RASMaskChangeGenerator` interface, which extends the `RASObject` interface. A short discussion of the methods that are introduced by each of these interfaces follows, along with implementation pointers. For more in-depth information on any of the particular interfaces or methods, see the corresponding product API documentation.

RASObject interface

The `RASObject` interface is the base interface for stand-alone JRas logging toolkit classes that are stateful or configurable, such as loggers, handlers, and formatters.

- The stand-alone JRas logging toolkit supports rudimentary properties-file based configuration. To implement this configuration support, the configuration state is stored as a set of key-value pairs in a properties file. The public `Hashtable getConfig` and public `void setConfig(Hashtable ht)` methods are used to get and set the configuration state. The JRas extensions do not support properties-based configuration. Implement these methods as no-operations. You can implement your own properties-based configuration using these methods.
- Loggers, handlers, and formatters can be named objects. For example, the JRas extensions require the user to provide a name for the loggers that are retrieved from the manager. You can name your handlers. The public `String getName` and public `void setName(String name)` methods are provided to get or set the name field. The JRas extensions currently do not call these methods on user handlers. You can implement these methods as you want, including as no operations.
- Loggers, handlers, and formatters can also contain a description field. The public `String getDescription` and public `void setDescription(String desc)` methods can be used to get or set the description field. The JRas extensions currently do not use the description field. You can implement these methods as you want, including as no operations.
- The public `String getGroup` method is provided for use by the `RASManager` interface. Since the JRas extensions provide their own `Manager` class, this method is never called. Implement this as a no-operation.

RASMaskChangeGenerator interface

The `RASMaskChangeGenerator` interface is the interface that defines the implementation methods for filtering of events based on a mask state. It is currently implemented by both loggers and handlers. By definition, an object that implements this interface contains both a message mask and a trace mask, although both need not be used. For example, message loggers contain a trace mask, but the trace mask is never used because the message logger never generates trace events. Handlers, however, can actively use both mask values. For example, a single handler can handle both message and trace events.

- The public `long getMessageMask` and public `void setMessageMask(long mask)` methods are used to get or set the value of the message mask. The public `long getTraceMask` and public `void setTraceMask(long mask)` methods are used to get or set the value of the trace mask.

In addition, this interface introduces the concept of *calling back* to interested parties when a mask changes state. The callback object must implement the `RASMaskChangeListener` interface.

- The public `void addMaskChangeListener(RASMaskChangeListener listener)` and public `void removeMaskChangeListener(RASMaskChangeListener listener)` methods are used to add or remove

listeners to the handler. The public Enumeration `getMaskChangeListeners` method returns an enumeration over the list of currently registered listeners. The public void `fireMaskChangedEvent(RASMaskChangeEvent mc)` method is used to call back all the registered listeners to inform them of a mask change event.

For efficiency reasons, the JRas extensions message and trace loggers implement the `RASIMaskChangeListener` interface. The logger implementations maintain a composite mask in addition to the logger mask. The logger composite mask is formed by logically *or'ing* the appropriate masks of all handlers that are registered to that logger, then *and'ing* the result with the logger mask. For example, the message logger composite mask is formed by or'ing the message masks of all handlers that are registered with that logger, then and'ing the result with the logger message mask.

All handlers are required to properly implement these methods. In addition, when a user handler is instantiated, the logger that is added must be registered with the handler; use the `addMaskChangeListener` method. When either the message mask or trace mask of the handler is changed, the logger must be called back to inform it of the mask change. With this process, the logger can dynamically maintain the composite mask.

The `RASMaskChangedEvent` class is defined by the stand-alone JRas logging toolkit. Direct use of that class by user code is supported in this context.

In addition, the `RASIMaskChangeGenerator` interface introduces the concept of caching the names of all message and trace event classes that the implementing object process. The intent of these methods is to support a management program such as a graphical user interface to retrieve the list of names, introspect the classes to determine the event types that they might possibly process and display the results. The JRas extensions do not ever call these methods, so they can be implemented as no operations.

- The public void `addMessageEventClass(String name)` and public void `removeMessageEventClass(String name)` methods can be called to add or remove a message event class name from the list. The method public Enumeration `getMessageEventClasses` returns an enumeration over the list of message event class names. Similarly, the public void `addTraceEventClass(String name)` and public void `removeTraceEventClass(String name)` methods can be called to add or remove a trace event class name from the list. The public Enumeration `getTraceEventClasses` method returns an enumeration over the list of trace event class names.

RASHandler interface

The `RASHandler` interface introduces the methods that are specific to the behavior of a handler.

The `RASHandler` interface, as provided by the stand-alone JRas logging toolkit, supports handlers that run in either a synchronous or asynchronous mode. In asynchronous mode, events are typically queued by the calling thread and then written by a worker thread. Because spawning of threads is not supported in the WebSphere Application Server environment, it is expected that handlers do not queue or batch events, although this activity is not expressly prohibited.

- The public int `getMaximumQueueSize()` and public void `setMaximumQueueSize(int size)` methods create `IllegalStateException` exceptions to manage the maximum queue size. The public int `getQueueSize` method is provided to query the actual queue size.
- The public int `getRetryInterval` and public void `setRetryInterval(int interval)` methods support the notion of error retry, which implies some type of queueing.
- The public void `addFormatter(RASIFormatter formatter)`, public void `removeFormatter(RASIFormatter formatter)` and public Enumeration `getFormatters` methods are provided to manage the list of formatters that the handler can be configured with. Different formatters can be provided for different event classes, if appropriate.
- The public void `openDevice`, public void `closeDevice` and public void `stop` methods are provided to manage the underlying device that the handler abstracts.
- The public void `logEvent(RASIEvent event)` and public void `writeEvent(RASIEvent event)` methods are provided to pass events to the handler for processing.

Writing a formatter

User-written formatters must implement the RASIFormatter interface. The RASIFormatter interface extends the RASIObjct interface. The implementation of the RASIObjct interface is the same for both handlers and formatters. A short discussion of the methods that are introduced by the RASIFormatter interface follows. For more in-depth information on the methods introduced by this interface, see the corresponding product API documentation.

RASIFormatter interface

- The public void setDefault(boolean flag) and public boolean isDefault methods are used by the concrete RASHandler classes that are provided by the stand-alone JRas logging toolkit to determine if a particular formatter is the default formatter. Because these RASHandler classes must never be used in a WebSphere Application Server environment, the semantic significance of these methods can be determined by the user.
- The public void addEventClass(String name), public void removeEventClass(String name) and public Enumeration getEventClasses methods are provided to determine which event classes a formatter can use to format. You can provide the appropriate implementations.
- The public String format(RASIEvent event) method is called by handler objects and returns a formatted String representation of the event.

Programming model summary

The programming model that is described in this section builds upon and summarizes some of the concepts already introduced. This section also formalizes usage requirements and restrictions. Use of the WebSphere Application Server JRas extensions in a manner that does not conform to the following programming guidelines is prohibited.

Note: The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

You can use the WebSphere Application Server JRas extensions in three distinct operational modes. The programming models concepts and restrictions apply equally across all modes of operation.

- You must not use implementation classes that are provided by the stand-alone JRas logging toolkit directly, unless specifically noted otherwise. Direct usage of those classes is not supported. IBM Support provides no diagnostic aid or bug fixes relating to the direct use of classes that are provided by the stand-alone JRas logging toolkit.
- You must obtain message and trace loggers directly from the Manager class. You cannot directly instantiate loggers.
- You cannot replace the WebSphere Application Server message and trace logger classes.
- You must guarantee that the logger names that are passed to the Manager class are unique, and follow the documented naming constraints. When a logger is obtained from the Manager class, you must not attempt to change the name of the logger by calling the setName method.
- Named loggers can be used more than once. For any given name, the first call to the Manager class results in the Manager class creating a logger that is associated with that name. Subsequent calls to the Manager class that specify the same name result in a returned reference to the existing logger.
- The Manager class maintains a hierarchical namespace for loggers. Use a dot-separated, fully qualified class name to identify any logger. Other than dots or periods, logger names cannot contain any punctuation characters, such as an asterisk (*), a comma (,), an equals sign (=), a colon (:), or quotes.
- Group names must comply with the same naming restrictions as logger names.
- The loggers returned from the Manager class are subclasses of the RASMessageLogger and the RASTraceLogger classes that are provided by the stand-alone JRas logging toolkit. You can call any public method that is defined by the RASMessageLogger and RASTraceLogger classes. You cannot call any public method that is introduced by the provided subclasses.
- If you want to operate in either stand-alone or combined mode, you must provide your own Handler and Formatter subclasses. You cannot use the Handler and Formatter classes that are provided by the stand-alone JRas logging toolkit. User written handlers and formatters must conform to the documented guidelines.

- Loggers that are obtained from the Manager class come with a WebSphere Application Server handler installed. This handler writes message and trace records to logs that are defined by the WebSphere Application Server run time. Manage these logs using the provided systems management interfaces.
- You can programmatically add and remove user-defined handlers from a logger at any time. Multiple additions and removals of user defined handlers are supported. You are responsible for creating an instance of the handler to add, configuring the handler by setting the handler mask value and formatter appropriately, then adding the handler to the logger using the addHandler method. You are responsible for programmatically updating the masks of user-defined handlers, as appropriate.
- You might get a reference to the handler that is installed within a logger by calling the getHandlers method on the logger and processing the results. You must not call any methods on the handler that are obtained in this way. You can remove the WebSphere Application Server handler from the logger by calling the logger removeHandler method, passing in the reference to the WebSphere Application Server handler. When removed, the WebSphere Application Server handler cannot be added again to the logger.
- You can define your own message type. The behavior of user-defined message types and restrictions on their definitions is discussed in JRas framework (deprecated).
- You can define your own message event classes. The use of user-defined message event classes is discussed in JRas framework (deprecated).
- You can define your own trace types. The behavior of user-defined trace types and restrictions on your definitions is discussed in JRas framework (deprecated).
- You can define your own trace event classes. The use of user-defined trace event classes is discussed in JRas framework (deprecated).
- You must programmatically maintain the bits in the message and trace logger masks that correspond to any user-defined types. If WebSphere Application Server facilities are used to manage the predefined types, these updates must not modify the state of any of the bits that correspond to those types. If you are assuming ownership responsibility for the predefined types, then you can change all bits of the masks.

JRas messages and trace event types

The basic JRas message and event types are not the same as those natively recognized by WebSphere Application Server, so the JRas types are mapped onto the types that are native to the runtime environment. You can control the way JRas message and trace events are processed using custom filters and message controls.

Event types

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The base message and trace event types that are defined by the stand-alone JRas logging toolkit are not the same as the native types that are recognized by the WebSphere Application Server run-time. Instead, the basic JRas types are mapped onto the native types. This mapping can vary by platform or edition. The mapping is discussed in the following section.

Platform message event types

The message event types that are recognized and processed by the WebSphere Application Server runtime are defined in the RASIMessageEvent interface that is provided by the stand-alone JRas logging toolkit.

Table 10. Platform message event types. These message types are mapped onto the native message types, as follows.

WebSphere Application Server native type	JRas RASIMessageEvent type
Audit	TYPE_INFO, TYPE_INFORMATION

Table 10. Platform message event types (continued). These message types are mapped onto the native message types, as follows.

WebSphere Application Server native type	JRas RASIMessageEvent type
Warning	TYPE_WARN, TYPE_WARNING
Error	TYPE_ERR, TYPE_ERROR

Platform trace event types

The trace event types that are recognized and processed by the WebSphere Application Server run time are defined in the RASITraceEvent interface that is provided by the stand-alone JRas logging toolkit. The RASITraceEvent interface provides a rich and complex set of types. This interface defines both a simple set of levels, as well as a set of enumerated types.

- For a user who prefers a simple set of levels, the RASITraceEvent interface provides TYPE_LEVEL1, TYPE_LEVEL2, and TYPE_LEVEL3. The implementations provide support for this set of levels. The levels are hierarchical, enabling level 2 also enables level 1, enabling level 3 also enables levels 1 and 2.
- For users who prefer a more complex set of values that can be *OR'd* together, the RASITraceEvent interface provides TYPE_API, TYPE_CALLBACK, TYPE_ENTRY_EXIT, TYPE_ERROR_EXC, TYPE_MISC_DATA, TYPE_OBJ_CREATE, TYPE_OBJ_DELETE, TYPE_PRIVATE, TYPE_PUBLIC, TYPE_STATIC, and TYPE_SVC.

The trace event types are mapped onto the native trace types as follows:

Table 11. WebSphere Application Server native types and JRas RASITraceEvent level types. Mapping WebSphere Application Server trace types to the JRas RASITraceEvent level types.

WebSphere Application Server native type	JRas RASITraceEvent level type
Event	TYPE_LEVEL1
EntryExit	TYPE_LEVEL2
Debug	TYPE_LEVEL3

Table 12. WebSphere Application Server native types and JRas RASITraceEvent enumerated types. Mapping WebSphere Application Server trace types to the JRas RASITraceEvent enumerated types.

WebSphere Application Server native type	JRas RASITraceEvent enumerated types
Event	TYPE_ERROR_EXC, TYPE_SVC, TYPE_OBJ_CREATE, TYPE_OBJ_DELETE
EntryExit	TYPE_ENTRY_EXIT, TYPE_API, TYPE_CALLBACK, TYPE_PRIVATE, TYPE_PUBLIC, TYPE_STATIC
Debug	TYPE_MISC_DATA

For simplicity, it is recommended that one or the other of the tracing type methodologies is used consistently throughout the application. If you decide to use the non-level types, choose one type from each category and use those types consistently throughout the application, to avoid confusion.

Message and trace parameters

The various message logging and trace method signatures accept the Object, Object[] and Throwable parameter types. WebSphere Application Server processes and formats the various parameter types as follows:

Primitives

Primitives, such as int and long are not recognized as subclasses of Object type and cannot be directly passed to one of these methods. A primitive value must be transformed to a proper Object type (Integer, Long) before passing as a parameter.

Object

The toString method is called on the object and the resulting String is displayed. Implement the toString method appropriately for any object that is passed to a message logging or trace method. It is the responsibility of the caller to guarantee that the toString method does not display confidential data such as passwords in clear text, and does not cause infinite recursion.

Object[]

The Object[] type is provided for the case when more than one parameter is passed to a message logging or trace method. The toString method is called on each Object in the array. Nested arrays are not handled, that is none of the elements in the Object array belong in an array.

Throwable

The stack trace of the Throwable type is retrieved and displayed.

Array of primitives

An array of primitive, for example, byte[], int[], is recognized as an Object, but is loosely associated by Java code. In general, avoid arrays of primitives, if possible. If arrays of primitives are passed, the results are indeterminate and can change, depending on the type of array passed, the API used to pass the array, and the release of the product. For consistent results, user code needs to preprocess and format the primitive array into some type of String form before passing it to the method. If such preprocessing is not performed, the following problems can result:

- [B@924586a0b - This message is deciphered as a byte array at location X. This message is typically returned when an array is passed as a member of an Object[] type and results from calling the toString method on the byte[] type.
- Illegal trace argument : array of long. This response is typically returned when an array of primitives is passed to a method taking an Object.
- 01040703: The hex representation of an array of bytes. Typically this problem can occur when a byte array is passed to a method taking a single Object. This behavior is subject to change and cannot be relied on.
- "1" "2": The String representation of the members of an int[] type formed by converting each element to an integer and calling the toString method on the integers. This behavior is subject to change and cannot be relied on.
- [Ljava.lang.Object;@9136fa0b : An array of objects. Typically this response is seen when an array containing nested arrays is passed.

Controlling message logging

Writing a message to a WebSphere Application Server log requires that the message type passes three levels of filtering or screening:

1. The message event type must be one of the message event types that is defined in the RASIMessageEvent interface.
2. Logging of that message event type must be enabled by the state of the message logger mask.
3. The message event type must pass any filtering criteria that is established by the WebSphere Application Server run-time.

When a WebSphere Application Server logger is obtained from the Manager class, the initial setting of the mask forwards all native message event types to the WebSphere Application Server handler. It is possible to control what messages get logged by programmatically setting the state of the message logger mask.

Some editions of the product support user specified message filter levels for a server process. When such a filter level is set, only messages at the specified severity levels are written to WebSphere Application Server. Message types that pass the mask check of the message logger can be filtered out by WebSphere Application Server.

Control tracing

Each edition of the product provides a mechanism for enabling or disabling trace. The various editions can support static trace enablement (trace settings are specified before the server is started), dynamic trace enablement (trace settings for a running server process can be dynamically modified), or both.

Writing a trace record to a WebSphere Application Server requires that the trace type passes three levels of filtering or screening:

1. The trace event type must be one of the trace event types that is defined in the RASITraceEvent interface.
2. Logging of that trace event type must be enabled by the state of the trace logger mask.
3. The trace event type must pass any filtering criteria that is established by the WebSphere Application Server run-time.

When a logger is obtained from the Manager class, the initial setting of the mask is to suppress all trace types. The exception to this rule is the case where the WebSphere Application Server run time supports static trace enablement and a non-default startup trace state for that trace logger is specified. Unlike message loggers, the WebSphere Application Server can dynamically modify the trace mask state of a trace logger. WebSphere Application Server only modifies the portion of the trace logger mask that corresponds to the values that are defined in the RASITraceEvent interface. WebSphere Application Server does not modify undefined bits of the mask that might be in use for user-defined types.

When the dynamic trace enablement feature that is available on some platforms is used, the trace state change is reflected both in the application server run time and the trace mask of the trace logger. If user code programmatically changes the bits in the trace mask corresponding to the values that are defined by in the RASITraceEvent interface, the mask state of the trace logger and the run time state become unsynchronized and unexpected results occur. Therefore, programmatically changing the bits of the mask corresponding to the values that are defined in the RASITraceEvent interface is not supported.

Instrumenting an application with JRas extensions

You can create an application using JRas extensions.

Before you begin

The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

To create an application using the WebSphere Application Server JRas extensions, perform the following steps:

Procedure

1. Determine the mode for the extensions: integrated, stand-alone, or combined.
2. If the extensions are used in either stand-alone or combined mode, create the necessary handler and formatter classes.
3. If localized messages are used by the application, create a resource bundle.
4. In the application code, get a reference to the Manager class and create the manager and logger instances.
5. Insert the appropriate message and trace logging statements in the application.

Creating JRas resource bundles and message files

The WebSphere Application Server message logger provides the message and msg methods so the user can log localized messages. In addition, the message logger provides the textMessage method to log messages that are not localized. Applications can use either or both, as appropriate.

Before you begin

The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

The mechanism for providing localized messages is the resource bundle support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with resource bundles as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the `java.util.ResourceBundle`, `java.util.ListResourceBundle` and `java.util.PropertyResourceBundle` classes, as well as the `java.text.MessageFormat` class.

The `PropertyResourceBundle` class is the preferred mechanism to use. In addition, note that the JRas extensions do not support the extended formatting options such as `{1, date}` or `{0, number, integer}` that are provided by the `MessageFormat` class.

You can forward messages that are written to the internal WebSphere Application Server logs to other processes for display. For example, messages that are displayed on the administrative console, which can be running in a different location than the server process, can be localized using the *late binding* process. Late binding means that WebSphere Application Server does not localize messages when they are logged, but defers localization to the process that displays the message.

To properly localize the message, the displaying process must have access to the resource bundle where the message text is stored. You must package the resource bundle separately from the application, and install it in a location where the viewing process can access it. If you do not want to take these steps, you can use the early binding technique to localize messages as they are logged.

The two techniques are described as follows:

Early binding

The application must localize the message before logging it. The application looks up the localized text in the resource bundle and formats the message. When formatting is complete, the application logs the message using the `textMessage` method. Use this technique to package the application resource bundles with the application.

Late binding

The application can choose to have the WebSphere Application Server run time localize the message in the process where it displays. Using this technique, the resource bundles are packaged in a stand-alone `.jar` file, separately from the application. You must then install the resource bundle `.jar` file on every machine in the installation from which an administrative console or log viewing program might be run. You must install the `.jar` file in a directory that is part of the extensions class path. In addition, if you forward logs to IBM service, you must also forward the `.jar` file that contains the resource bundles.

To create a resource bundle, perform the following steps.

Procedure

1. Create a text properties file that lists message keys and the corresponding messages. The properties file must have the following characteristics:
 - Each property in the file is terminated with a line-termination character.
 - If a line contains only white space, or if the first non-white space character of the line is the number sign symbol (`#`) or exclamation mark (`!`), the line is ignored. The `#` and `!` characters can therefore be used to put comments into the file.
 - Each line in the file, unless it is a comment or consists only of white space, denotes a single property. A backslash (`\`) is treated as the line-continuation character.
 - The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (`=`), colon (`:`), and white space ().
 - The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (`\`), but using this approach is not recommended because escaping characters is error prone and confusing. Instead, use a valid separator character that does not display in any keys in the properties file.

- White space after the key and separator is ignored until the first non-white space character is encountered. All characters that remain before the line-termination character define the element. See the Java documentation for the `java.util.Properties` class for a full description of the syntax and construction of properties files.
2. Translate the file into localized versions of the file with language-specific file names for example, the `DefaultMessages.properties` file can be translated into `DefaultMessages_de.properties` for German and `DefaultMessages_ja.properties` for Japanese.
 3. When the translated resource bundles are available, write them to a system-managed persistent storage medium. Resource bundles are used to convert the messages into the requested national language and locale.
 4. When a message logger is obtained from the JRes manager, configure the logger to use a particular resource bundle. Messages logged through the message API use this resource bundle when message localization is performed. At run time, the user's locale setting is used to determine the properties file from which to extract the message that is specified by a message key, ensuring that the message is delivered in the correct language.
 5. If the message loggers `msg` method is called, explicitly identify a resource bundle name.

What to do next

The application locates the resource bundle based on the file location relative to any directory in the class path. For instance, if the `DefaultMessages.properties` property resource bundle is in the `baseDir/subDir1/subDir2/resources` directory and `baseDir` is in the class path, the name `subdir1.subdir2.resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

JRes resource bundles:

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a `PropertiesResourceBundle` resource bundle. This sample shows how to create such a properties file.

Resource bundle sample

The JRes framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs are inserted into it. All the normal properties files conventions and rules apply to this file. In addition, the creator must be aware of other restrictions that are imposed on the values by the Java `MessageFormat` class. For example, apostrophes must be escaped or they cause a problem. Avoid the use of non-portable characters. WebSphere Application Server does not support the use of extended formatting conventions that the `MessageFormat` class supports, such as `{1, date}` or `{0, number, integer}`.

Assume that the base directory for the application that uses this resource bundle is `baseDir` and that this directory is in the class path. Assume that the properties file is stored in the subdirectory `baseDir` that is not in the class path (`baseDir/subDir1/subDir2/resources`). To allow the messages file to resolve, the `subDir1.subDir2.resources.DefaultMessage` name is used to identify the `PropertyResourceBundle` resource bundle and is passed to the message logger.

For this sample, the properties file is named `DefaultMessages.properties`:

```
# Contents of the DefaultMessages.properties file
MSG_KEY_00=A message with no substitution parameters.
MSG_KEY_01=A message with one substitution parameter: parm1={0}
MSG_KEY_02=A message with two substitution parameters: parm1={0}, parm2 = {1}
MSG_KEY_03=A message with three substitution parameters: parm1={0}, parm2 = {1}, parm3={2}
```


When the `DefaultMessages.properties` file is created, the file can be sent to a translation center where the localized versions are generated.

JRas manager and logger instances

You can use the JRas extensions in integrated, stand-alone, or combined mode. Configuration of the application varies depending on the mode of operation, but use of the loggers to log message or trace entries is identical in all modes of operation.

Deprecated: The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

Integrated mode is the default mode of operation. In this mode, message and trace events are sent to the WebSphere Application Server logs.

In the combined mode, message and trace events are logged to both WebSphere Application Server and user-defined logs.

In the stand-alone mode, message and trace events are logged only to user-defined logs.

Using the message and trace loggers

Regardless of the mode of operation, the use of message and trace loggers is the same.

Using a message logger

The message logger is configured to use the `DefaultMessages` resource bundle. Message keys must be passed to the message loggers if the loggers are using the message API.

```
msgLogger.message(RASIMessageEvent.TYPE_WARNING, this,
    methodName, "MSG_KEY_00");
... msgLogger.message(RASIMessageEvent.TYPE_WARN, this,
    methodName, "MSG_KEY_01", "some string");
```

If message loggers use the `msg` API, you can specify a new resource bundle name.

```
msgLogger.msg(RASIMessageEvent.TYPE_ERR, this, methodName,
    "ALT_MSG_KEY_00", "alternateMessageFile");
```

You can also log a text message. If you are using the `textMessage` API, no message formatting is done.

```
msgLogger.textMessage(RASIMessageEvent.TYPE_INFO, this, methodName, "String and Integer",
    "A String", new Integer(5));
```

Using a trace logger

Because trace is normally disabled, guard trace methods for performance reasons.

```
private void methodX(int x, String y, Foo z)
{
    // trace an entry point. Use the guard to make sure tracing is enabled.
    Do this checking before you gather parameters to trace.
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT) {
        // I want to trace three parameters, package them up in an Object[]
        Object[] parms = {new Integer(x), y, z};
        trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX", parms);
    }
    ... logic
    // a debug or verbose trace point
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA) {
        trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA, this, "methodX" "reached here");
    }
    ...
    // Another classification of trace event. An important state change is
```

```

detected, so a different trace type is used.
if (trcLogger.isLoggable(RASITraceEvent.TYPE_SVC) {
    trcLogger.trace(RASITraceEvent.TYPE_SVC, this, "methodX", "an important event");
}
...
// ready to exit method, trace. No return value to trace
if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT)) {
    trcLogger.exit(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX");
}
}

```

Setting up for integrated JRas operation

Use JRas operations in integrated mode to send trace events and logging messages to only WebSphere Application Server logs.

Before you begin

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

In the integrated mode of operation, message and trace events are sent to WebSphere Application Server logs. This approach is the default mode of operation.

Procedure

1. Import the requisite JRas extensions classes:

```

import com.ibm.ras.*;
import com.ibm.websphere.ras.*;

```

2. Declare logger references:

```

private RASMessageLogger msgLogger = null;
private RASTraceLogger trcLogger = null;

```

3. Obtain a reference to the Manager class and create the loggers. Because loggers are named singletons, you can do this activity in a variety of places. One logical candidate for enterprise beans is the `ejbCreate` method. For example, for the `myTestBean` enterprise bean, place the following code in the `ejbCreate` method:

```

com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
    myTestBean.class.getName());

```

```

// Configure the message logger to use the message file that is created
// for this application.
msgLogger.setMessageFile("acme.widgets.DefaultMessages");
trcLogger = mgr.createRASTraceLogger("Acme", "Widgets", "RasTest",
    myTestBean.class.getName());
mgr.addLoggerToGroup(trcLogger, groupName);

```

Setting up for combined JRas operation

Use JRas operation in combined mode to output trace data and logging messages to both WebSphere Application Server and user-defined logs.

Before you begin

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

In combined mode, messages and trace are logged to both WebSphere Application Server logs and user-defined logs. The following sample assumes that:

- You wrote a user-defined handler named SimpleFileHandler and a user-defined formatter named SimpleFormatter.
- You are not using user-defined types or events.

Procedure

1. Import the requisite JRas extensions classes:

```
import com.ibm.ras.*;
import com.ibm.websphere.ras.*;
```

2. Import the user handler and formatter:

```
import com.ibm.ws.ras.test.user.*;
```

3. Declare the logger references:

```
private RASMessageLogger msgLogger = null;
private RASTraceLogger trcLogger = null;
```

4. Obtain a reference to the Manager class, create the loggers, and add the user handlers. Because loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the `ejbCreate` method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must support this approach. The following sample is a message logger sample. The procedure for a trace logger is similar.

```
com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
    myTestBean.class.getName());
// Configure the message logger to use the message file defined
// in the ResourceBundle sample.
msgLogger.setMessageFile("acme.widgets.DefaultMessages");

// Create the user handler and formatter. Configure the formatter,
// then add it to the handler.
RASHandler handler = new SimpleFileHandler("myHandler", "FileName");
RASFormatter formatter = new SimpleFormatter("simple formatter");
formatter.addEventClass("com.ibm.ras.RASMessageEvent");
handler.addFormatter(formatter);

// Add the Handler to the logger. Add the logger to the list of the
//handlers listeners, then set the handlers
// mask, which updates the loggers composite mask appropriately.
// WARNING - there is an order dependency here that must be followed.
msgLogger.addHandler(handler);
handler.addMaskChangeListener(msgLogger);
handler.setMessageMask(RASMessageEvent.DEFAULT_MESSAGE_MASK);
```

Setting up for stand-alone JRas operation

You can configure JRas operations to output trace data and logging messages to only user-defined locations.

Before you begin

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

About this task

In stand-alone mode, messages and traces are logged only to user-defined logs. The following sample assumes that:

- You have a user-defined handler named SimpleFileHandler and a user-defined formatter named SimpleFormatter.
- You are not using user-defined types of events.

Procedure

1. Import the requisite JRas extensions classes:

```
import com.ibm.ras.*;
import com.ibm.websphere.ras.*;
```

2. Import the user handler and formatter:

```
import com.ibm.ws.ras.test.user.*;
```

3. Declare the logger references:

```
private RASMessageLogger msgLogger = null;
private RASTraceLogger trcLogger = null;
```

4. Obtain a reference to the Manager class, create the loggers, and add the user handlers. Because loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the `ejbCreate` method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must support this approach. The following sample is a message logger sample. The procedure for a trace logger is similar.

```
com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
    myTestBean.class.getName());
// Configure the message logger to use the message file that is defined in
//the ResourceBundle sample.
msgLogger.setMessageFile("acme.widgets.DefaultMessages");

// Get a reference to the Handler and remove it from the logger.
RASHandler aHandler = null;
Enumeration enum = msgLogger.getHandlers();
while (enum.hasMoreElements()) {
    aHandler = (RASHandler)enum.nextElement();
    if (aHandler instanceof WsHandler)
        msgLogger.removeHandler(wsHandler);
}

// Create the user handler and formatter. Configure the formatter,
// then add it to the handler.
RASHandler handler = new SimpleFileHandler("myHandler", "FileName");
RASFormatter formatter = new SimpleFormatter("simple formatter");
formatter.addEventClass("com.ibm.ras.RASMessageEvent");
handler.addFormatter(formatter);

// Add the Handler to the logger. Add the logger to the list of the
// handlers listeners, then set the handlers
// mask, which will update the loggers composite mask appropriately.
// WARNING - there is an order dependency here that must be followed.
msgLogger.addHandler(handler);
handler.addMaskChangeListener(msgLogger);
handler.setMessageMask(RASMessageEvent.DEFAULT_MESSAGE_MASK);
```

Logging Common Base Events in WebSphere Application Server

WebSphere Application Server uses Common Base Events within its basic logging framework. Common Base Events can be created explicitly and then logged through the Java logging API, or can be created implicitly by using the Java logging API directly.

About this task

Attention: Logging Common Base Events is not supported with the High Performance Extensible Logging (HPEL) log and trace mode.

An *event* is a notification from an application or the application server that reports information that is related to a specific problem or situation. Common Base Events provide you with a standard structure for these event notifications, which allow you to correlate events that are received from different applications. Log Common Base Events to capture events from different sources to help you fix a problem within an application environment or to tune system performance.

For Common Base Event creation, the application server environment provides a Common Base Event factory with a content handler that provides both runtime data and template data for Common Base Events.

Procedure

1. Optional: Read about the Common Base Event types and how they are implemented within an application server. Refer to “The Common Base Event in WebSphere Application Server.”
2. Read “Logging Common Base Events in WebSphere Application Server” on page 100.
3. Configure the Common Base Event framework for your application server using one of the following methods:
 - “Logging with Common Base Event API and the Java logging API” on page 88
 - “Generate Common Base Event content with the default event factory” on page 90.

Results

Common Base Events will now be logged according to your configuration. Use these event logs to determine the source of application problems.

The Common Base Event in WebSphere Application Server

The Common Base Event is an XML document that defines a common representation of events that is intended for use by enterprise management and business applications. The Common Base Event defines common fields, the values they can take, and the exact meanings of these values.

An application creates an event object whenever something happens that either needs to be recorded for later analysis or which might require the trigger of additional work. An *event* is a structured notification that reports information that is related to a situation. An event reports three kinds of information:

- The situation: What happened
- The identity of the affected component: For example, the server that shut down
- The identity of the component that is reporting the situation, which might be the same as the affected component

The application that creates the event object is called the *event source*. Event sources can use a common structure for the event. The accepted standard for such a structure is called the *Common Base Event*. The Common Base Event is an XML document that is defined as part of the autonomic computing initiative.

The Common Base Event model is a standard that defines a common representation of events that is intended for use by enterprise management and business applications. This standard, which is developed by the IBM Autonomic Computing Architecture Board, supports encoding of logging, tracing, management, and business events using a common XML-based format. This format makes it possible to correlate different types of events that originate from different applications. For more information about the Common Base Event model, see the Common Base Event specification (*Canonical Situation Data Format: The Common Base Event V1.0.1*). The common event infrastructure currently supports Version 1.0.1 of the specification.

Note:

For WebSphere Application Server Version 8.5, if you delete an application server that was previously deployed with the Common Event Infrastructure (CEI) enabled and you did not uninstall CEI before deleting the server, you must use a different name when creating an application server that you want to deploy with CEI. If you deploy CEI on an application server that was created with the exact same server name as the server that was previously deleted and CEI was not uninstalled, the following error occurs:

```
com.ibm.websphere.management.exception.AdminException: ADMA5026E: No valid target is specified in ObjectName  
WebSphere:cell=targetCell,node=targetNode,server=targetServer for module EventServerMdb.jar+META-INF/ejb-jar.xml
```

If you did not uninstall CEI before deleting the application server, you must ensure that you use a name for the new application server that is different from the name of the server that was previously deployed with the common event infrastructure.

The basic concept behind the Common Base Event model is the *situation*. A situation can be anything that happens anywhere in the computing infrastructure, such as a server shutdown, a disk-drive failure, or a failed user login. The Common Base Event model defines a set of standard situation types that accommodate most of the situations that might arise (for example, StartSituation and CreateSituation).

The Common Base Event contains all of the information that is needed by the consumers to understand the event. This information includes data about the runtime environment, the business environment, and the instance of the application object that created the event.

For complete details on the Common Base Event format, see the XML schema that is included in the Common Base Event specification document, at <http://www.ibm.com/developerworks/autonomic/books/fpy0mst.htm#HDRCBEDDESC> .

Types of problem determination events

Problem determination involves multiple types of data, including at least two different classes of event data, log events, and diagnostic events.

Log events, which are also referred to as *message events*, are typically emitted by components of a business application during normal deployment and operations. Log events might identify problems, but these events are also normally available and emitted while an application and its components are in production mode. The target audience for log and message events is users and administrators of the application and the components that make up the application. Log events are normally the only events available when a problem is first detected, and are typically used during both problem recovery and problem resolution.

Diagnostic events, which are commonly referred to as *trace events*, are used to capture internal diagnostic information about a component, and are usually not emitted or available during normal deployment and operation. The target audience for diagnostic events is the developers of the components that make up the business application. Diagnostic events are typically used when trying to resolve problems within a component, such as a software failure, but are sometimes used to diagnose other problems, especially when the information provided by the log events is not sufficient to resolve the problem. Diagnostic events are typically used when trying to resolve a problem.

A *Common Base Event* is a common structure for an event. It defines common fields, the values that these fields can take, and the exact meanings of these values for an event. Common Base Events are primarily used to represent log events.

Common Base Event structure

A *Common Base Event* is a common structure for an event. It defines common fields, the values that these fields can take, and the exact meanings of these values for an event.

The Common Base Event contains several structural elements. These elements include:

- Common header information
- Component identification, both source and reporter
- Situation information
- Message data
- Extended data
- Context data
- Associated events and association engine

Each of these structural elements has its own embedded elements and attributes.

The following table presents a summary of all the fields in the Common Base Event and their usage requirements for problem determination events.

Table 13. Field name, log events, and base specification. This table shows whether a particular element or attribute is required, recommended, optional, prohibited, or discouraged for log events, and the base specification.

Field name	Log events	Base specification
Version	Required	Required
creationTime	Required	Required
severity	Required	Optional
Msg	Required	Optional
sourceComponentId*	Required	Required
sourceComponentId.location	Required	Required
sourceComponentId.locationType	Required	Required
sourceComponentId.component	Required	Required
sourceComponentId.subComponent	Required	Required
sourceComponentId.componentIdType	Required	Required
sourceComponentId.componentType	Required	Required
sourceComponentId.application	Recommended	Optional
sourceComponentId.instanceId	Recommended	Optional
sourceComponentId.processId	Recommended	Optional
sourceComponentId.threadId	Recommended	Optional
sourceComponentId.executionEnvironment	Optional	Optional
situation*	Required	Required
situation.categoryName	Required	Required
situation.situationType*	Required	Required
situation.situationType.reasoningScope	Required	Required
situation.situationType.(specific Situation Type elements)	Required	Required
msgDataElement*	Recommended	Optional
msgDataElement .msgId	Recommended	Optional
msgDataElement .msgIdType	Recommended	Optional
msgDataElement .msgCatalogId	Recommended	Optional
msgDataElement .msgCatalogTokens	Recommended	Optional
msgDataElement .msgCatalog	Recommended	Optional
msgDataElement .msgCatalogType	Recommended	Optional

Table 13. Field name, log events, and base specification (continued). This table shows whether a particular element or attribute is required, recommended, optional, prohibited, or discouraged for log events, and the base specification.

Field name	Log events	Base specification
msgDataElement .msgLocale	Recommended	Optional
extensionName	Recommended	Optional
localInstanceid	Optional	Optional
globalInstanceid	Optional	Optional
priority	Discouraged	Optional
repeatCount	Optional	Optional
elapsedTime	Optional	Optional
sequenceNumber	Optional	Optional
reporterComponentId*	Optional	Optional
reporterComponentId.location	Required (2)	Required (2)
reporterComponentId.locationType	Required (2)	Required (2)
reporterComponentId.component	Required (2)	Required (2)
reporterComponentId.subComponent	Required (2)	Required (2)
reporterComponentId.componentIdType	Required (2)	Required (2)
reporterComponentId.componentType	Required (2)	Required (2)
reporterComponentId.instanceid	Optional	Optional
reporterComponentId.processId	Optional	Optional
reporterComponentId.threadId	Optional	Optional
reporterComponentId.application	Optional	Optional
reporterComponentId.executionEnvironment	Optional	Optional
extendedDataElements*	Note 3	Optional
contextDataElements*	Note 4	Optional
associatedEvents*	Note 5	Optional

Notes:

- Items followed by an asterisk (*) are elements that consist of sub elements and attributes. The fields in those elements are listed in the table directly following the parent element name.
- Some of the elements are optional, but when included, they include sub elements and attributes that are required. For example, the reporterComponentId element has a ComponentIdentification type. The component attribute in ComponentIdentification is required. Therefore, the reporterComponentId.component attribute is required, but only when the reporterComponentId parent element is included.
- The extendedDataElements element can be included multiple times to supply extended data information. See the Extended data section for more information on required and recommended extended data element values.
- The contextDataElements element can be included multiple times to supply context data information.
- The associatedEvents element can be included multiple times to supply correlation data. No recommended uses of this element exist for the producers of problem determination data, and the use of this element is discouraged.

Common header information:

This topic provides additional information about how to format and use these fields for problem determination events, which can be used to clarify and extend the information provided in the other documents.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

The common header information in the Common Base Event includes the following information about an event:

- Version: The version of this Common Base Event
- creationTime: The date and time when the event generated
- Severity and priority: The severity of the condition (situation) that is identified by the event
- extensionName: The type of event that was captured
- localInstanceId and globalInstanceId: Identifiers that can be used to quickly identify a specific event within a set of events
- repeatCount and elapsedTime: Information that supports a system to efficiently report multiple events of the same type, by consolidating those events into a single event
- sequenceNumber: Sequence information that supports a system to order a set of events in other ways than time of capture

severity

All problem determination events must provide an indication as to the relative severity of the condition (situation) being reported by providing appropriate values for the severity field in the Common Base Event. The severity field is required for problem determination events. This field is more restrictive than the base specification for the Common Base Event, which lists this field as optional because effective and efficient problem determination requires the ability to quickly identify the information that is needed to resolve a problem as well as prioritize the problems that need addressing.

Table 14. Severity values. The following values are used for problem determination events:

Value	Severity	Description
10	Information	Log information events, normal conditions, and events that are supplied to clarify operations, for example, state transitions, operational changes. These events typically do not require administrator action or intervention.
20	Harmless	Similar to information events, but are used to capture audit items, such as state transitions or operational changes. These events typically do not require administrator action or intervention.
30	Warning	Warnings typically represent recoverable errors, for example a failure that the system can correct. These events can require administrator action or intervention.

Table 14. Severity values (continued). The following values are used for problem determination events:

Value	Severity	Description
40	Minor	Minor errors describe events that represent an unrecoverable error within a component. The failure affects the component ability to service some requests. The business application can continue to perform its normal functions, but its overall operation might be degraded. These events require administrator action or intervention to address the condition.
50	Critical	Critical errors describe events that represent an unrecoverable error within a component. The failure significantly affects the component ability to service most requests. The business application can continue most, but not all of its normal functions and its overall operation might be degraded. These events require administrator action or intervention to address the condition.
60	Fatal	Fatal errors describe events that represent an unrecoverable error within a component. The failure usually results in the complete failure of the component. The business application can continue some normal functions, but its overall operation might be degraded. These events require administrator action or intervention to address the condition.

msg

Refer to “Message data” on page 84 for information on this attribute.

priority

The use of the priority field is discouraged for problem determination events. The severity field is typically used to communicate and evaluate the importance of problem determination events. Use the priority field to enhance the information that is provided in the severity field, that is, prioritize events of the same severity.

extensionName

The extensionName field is used to communicate the type of event that is reported, for example, what general class of events is being reported. In many cases this field provides an indication of what additional data you can expect with the event, for example, optional data values.

repeatCount

The repeatCount field is valid for problem determination events, but is not typically used or supplied by the event producers. This field is used for data reduction and consolidation by event management and analysis systems.

elapsedTime

The elapsedTime field is valid for problem determination events, but is not typically used or supplied by the event producers. This field is used for data reduction and consolidation by event management and analysis systems.

sequenceNumber

The sequenceNumber field is valid for problem determination events. It is typically used only by event producers when the granularity of the event time stamp (the creationTime field) is not sufficient in ordering events. The sequenceNumber field is typically used to sequence events that have the same time stamp value.

Event management and analysis systems can use the sequenceNumber field for a number of reasons, including providing alternative sequencing, not necessarily based on a time stamp. The recommendations here are provided primarily for event producers.

Component identification for source and reporter:

The component identification fields in the Common Base Event are used to indicate which component in the system is experiencing the condition that is described by the event (the sourceComponentID) and which component emitted the event (the reporterComponentID).

Typically, these components are the same, in which case only the sourceComponentID is supplied. Some notes and scenarios on when to use these two elements in the Common Base Event:

- The sourceComponentID is always used to identify the component experiencing the condition that is described by the event.
- The reporterComponentID is used to identify the component that actually produced and emitted the event. This element is typically used only within events that are emitted by a component that is monitoring another component and providing operational information regarding that component. The monitoring component (for example, a Tivoli agent or hardware device driver) is identified by the reporterComponentID and the component being monitored (for example, a monitored server or hardware device) is identified by the sourceComponentID.

A potential misuse of the reporterComponentID is to identify a component that provides event conversion or management services for a component, for example, identifying an adapter that transforms the events that are captured by a component into Common Base Event format. The event conversion function is considered an extension of the component and not identified separately.

The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component.

Table 15. Component identification for source and reporter. The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component.

Source component	Reporter component	Description
location locationType	Component location	Identifies the location of the component.
component componentIdType	Component name	Identifies the asset name of the component, as well as the type of component.
subcomponent	Subcomponent name	Identifies a specific part or subcomponent of a component, for example a software module or hardware part.
application	Business application name	Identifies the business application or process the component is a part of and provides services for.
instanceId	Operational instance	Identifies the operational instance of a component, that is the actual running instance of the component.

Table 15. Component identification for source and reporter (continued). The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component.

Source component	Reporter component	Description
processId threadId	Operational instance	Identifies the operational instance of a component within the context of a software operating system, that is the operating system process and thread running when the event was produced.
executionEnvironment	Operational instance Component location	Provides additional information about the operational instance of a component or its location by identifying the name of the environment hosting the operational instance of the component, for example the operating system name for a software application, the application server name for a Java 2 Platform, Enterprise Edition (J2EE) application, or the hardware server type for a hardware part.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines. This section provides additional information about how to format and use some of these fields for problem determination events, which can be used to clarify and extend the information that is provided in the other documents.

Component

The Component field in a problem determination event is used to identify the manageable asset that is associated with the event. A manageable asset is open for interpretation, but a good working definition is a manageable asset represents a hardware or software component that can be separately obtained or developed, deployed, managed, and serviced. Examples of typical component names are:

- IBM eServer xSeries® model x330
- IBM WebSphere Application Server version 5.1 (5.1 is the version number)
- The name of an internally developed software application for a component

subComponent

The Subcomponent field in a problem determination event identifies the specific part of a component that is associated with the event. The subcomponent name is typically not a manageable asset, but provides internal diagnostic information when diagnosing an internal defect within a component, that is What part failed? Examples of typical subcomponents and their names are:

- Intel Pentium processor within a server system (Intel Pentium IV Processor)
- the enterprise bean container within a web application server (enterprise bean container)
- the task manager within an operating system (Linux Kernel Task Manager)
- the name of a Java class and method (myclass.mycompany.com or myclass.mycompany.com.methodname).

The format of a subcomponent name is determined by the component, but use the convention shown previously for naming a Java class or the combination of a Java class and method is followed. The subcomponent field is required in the Common Base Event.

componentIdType

The componentIdType field is required by the Common Base Event specification, but provides minimal value for problem determination events. For most problem determination events, it is encouraged to

use the value provided in the application field instead of the componentIdType. The componentIdType field identifies the type of component; the application is identified by the application field.

application

The application field is listed as an optional value within the Common Base Event specification, but provide it within problem determination events whenever it this value is available. The only reason this field is not required for problem determination events is that instances exist where the issuing component might not be aware of the overall business application.

instanceId

The instanceId field is listed as an optional value within the Common Base Event specification, but provide this value within problem determination events whenever it is available.

Always provide the instanceId when a software component is identified and identify the operational instance of the component (for example, which operation instance of an installed software image is actually associated with the event). Provide this value for hardware components when these components support the concept of operational instances.

The format of the supplied value is defined by the component, but must be a value that an analysis system can use (either human or programmatic) to identify the specific running instance of the identified component. Examples include:

- **cell, node, server** name for the IBM WebSphere Application Server
- **deployed EAR file name** for a Java enterprise bean
- **serial number** for a hardware processor

processId

The processId field is listed as an optional value within the Common Base Event specification, but provide this value for problem determination events whenever it is available and applicable. Always provide this value for software-generated events, and identify the operating system process that is associated with the component that is identified in the event. Match the format of the thread ID with the format of the operating system (or other running environment, such as a Java virtual machine). This field is typically not applicable or used for events that are emitted by hardware (for example, firmware).

threadId

The threadId field is listed as an optional value within the Common Base Event specification, but provide this value for problem determination events whenever it is available and applicable. Always provide for software-generated events, and identify the active operating system thread when the event was detected or issued. A notable exception to this recommendation is some operating systems or running environments do not support threads. Match the format of the thread ID with the format of the operating system (or other running environment, such as a Java virtual machine). This field is typically not applicable or used for events that are emitted by hardware (for example, firmware).

executionEnvironment

The executionEnvironment field, when used, identifies the immediate running environment that is used by the component being identified. Some examples are:

- the operating system name when the component is a native software application.
- the operating system/Java virtual machine name when the component is a Java 2 Platform, Standard Edition (J2SE) application.
- the web server name when the component is a servlet.
- the portal server name when the component is a portlet.
- the application server name when the component is an enterprise bean.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

Situation information:

The situation information is used to classify the condition that is reported by an event into a common set of situations.

The Common Base Event specification [CBE101] provides information on the set of situations defined for the Common Base Event, with the values and formats that are used to describe these situations. The Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

Consider the following points regarding situation information for problem determination events:

- Whenever possible, use the situation categorizations and qualifiers that are described in the base Common Base Event specification. Avoid using your own situation definitions as much as possible.
- Not all messages and logs can be classified using the situation definitions that are supplied in the base Common Base Event specification. You can use the OtherSituation categorization to provide your own situation information, but the recommended course of action for problem determination events is to use the ReportSituation categorization, with reportCategory=Log.
- Warning events can be confusing. A warning event (that is an event with severity=warning) typically indicates a recoverable failure, but the situation settings can be interpreted as unrecoverable failures (for example ConnectSituation, successDisposition=UNSUCCESSFUL). Use the appropriate situation categorization so the severity setting indicates the severity of the situation, that is whether the component recovered from the failure.
- The recommended setting for the reasoningScope value is EXTERNAL for all message events.

Message data:

All problem determination Common Base Events must provide human readable text that describes the specific reported event within the msg field of the Common Base Event.

The text that is associated with events representing actual messages or log entries is expected to be translated and localized. Include the msgDataElement element in the Common Base Event whenever internationalized text is provided in the event. This element provides information about how the message text is created and how to interpret it. This information is particularly invaluable when trying to interpret the event programmatically or when trying to interpret the message independent of the locale or language that is used to format the message text.

Prerequisite: Understand the concepts that are associated with creating internationalized messages. A good source of education on these concepts is provided by the documentation that is associated with internationalization of Java information and the usage of resource bundles within the Java language.

The msgDataElement element in the Common Base Event includes the following information about the value of the msg field that is provided with an event:

- The locale of the supplied message text, which identifies how the locale-independent fields within the message are formatted, as well as the language of the message (msgLocale).
- A locale-independent identifier that is associated with the message that can be used to interpret the message independent of the message language, message locale, and message format (msgId and msgIdType).
- Information on how a translated message is created, including:
 - The identifier that is used to retrieve the message template (msgCatalogId).
 - The name and type of message catalog that are used to retrieve the message template (msgCatalog and msgCatalogType).
 - Any locale-independent information that is inserted into the message template to create the final message (msgCatalogTokens).

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines. This section provides additional information about how to format and use these fields for problem determination events.

msg

All message, log, and trace events must provide a human-readable message in the msg field of the Common Base Event. The msg field is required for problem determination events, both log events and diagnostic events. This field is more restrictive than the base specification for the Common Base Event, which lists this field as optional; effective and efficient problem determination requires the ability to quickly identify the reported condition. The format and usage of this message is component-specific, but use the following general guidelines:

- Expect the message text that is supplied with messages and log events to be internationalized.
- Provide the locale of the supplied message text with the msgLocale field in the msgDataElement element of the Common Base Event.
- Provide additional information regarding the format and construction of internationalized messages whenever possible, using the msgDataElement element of the Common Base Event.

msgLocale

Provide the message locale whenever message text is provided within the Common Base Event, as is the case with all problem determination events. The msgLocale field is listed as an optional value within the Common Base Event specification, but provide this information within problem determination events whenever possible. The reason this field is not required for problem determination events is that instances exist where the locale information is not provided or available when formatting the Common Base Event.

msgId and msgIdType

Several companies include a locale-independent identifier within internationalized message text that you can use to interpret the described condition by the message text, independent of the message. For example, most messages issued by IBM software look like IEE890I WTO Buffers in console backup storage = 1024, where a unique, locale-independent identifier IEE890I precedes the translated message text. This identifier provides a way to uniquely detect and identify a message independent of location and language. This detection is invaluable for locale-independent and programmatic analysis.

The msgId field is listed as an optional value within the Common Base Event specification, but it must be provided within problem determination events whenever this identifier is included in the message text. Likewise, the msgIdType field is listed as an optional value within the Common Base Event specification, but it must be provided within problem determination events whenever a value is supplied for msgId. Do not supply these fields when the message text is not translated or localized, for example, for trace events.

msgCatalogId

The msgCatalogId field is listed as an optional value within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. This field is not required for problem determination events because not all problem determination events include translated message text. Some cases exist where the value is not provided or available when formatting the Common Base Event. Do not supply this field when the message text is not translated or localized, for example, for trace events.

msgCatalogTokens

The msgCatalogTokens field is listed as an optional value within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. This field is not required for problem determination events because not all problem determination events include translated message text, and cases exist where the value is not

provided or available when formatting the Common Base Event. This value contains the list of locale-independent values or message tokens that are inserted into the localized message text when creating a translated message.

These values are difficult to extract from a translated message without knowing the translated message template that is used to create the message. Do not supply this field when the message text is not translated or localized

The Common Base Event provides several mechanisms for providing additional data about an event, including this field, extended data elements, and extensions to the schema. Always use the `msgCatalogTokens` field to supply the list of message tokens that is included in the message text associated with an event. These values can also be supplied in other parts of the Common Base Event, but they must be included in this field.

msgCatalog and msgCatalogType

The `msgCatalog` and `msgCatalogType` fields are listed as optional values within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. These fields are not required for problem determination events because not all problem determination events include translated message text, and cases exist where the values are not provided or available when formatting the Common Base Event. Do not complete these fields when the message text has is not translated or localized, for example, for trace events.

Extended data:

The Common Base Event provides several methods for including this additional data, including extending the Common Base Event schema or supplying one or more `ExtendedDataElement` elements within the Common Base Event, which is the preferred approach.

The base information that is included in a Common Base Event might not be sufficient to represent all of the information captured by a component when creating a problem determination event.

Use an `ExtendedDataElement` element to represent a single data item. A Common Base Event can contain more than one of these elements, essentially one for each additional data item. A hint to the number and type of `ExtendedDataElement` elements is supplied by the `extensionName` value, but this information is only a hint. The usage of the attributes in the `ExtendedDataElement` element for problem determination events is the same as those for any other Common Base Event.

Sample Common Base Event instance

This XML document is an example of a Common Base Event instance that is generated by a WebSphere Application Server application.

Use the following example for reference:

```
<CommonBaseEvent creationTime="2004-09-18T04:03:28.484Z"
  globalInstanceId="myhost:1095479647062:1899"
  msg="WSVR0024I: Server server1 stopped"
  severity="10"
  version="1.0.1">
  ... several extendedDataElements for internal use only ...
<sourceComponentId component="com.ibm.ws.runtime.component.ServerCollaborator"
  componentIdType="Unknown"
  executionEnvironment="Windows Vista[x86]#5.0"
  instanceId="myhost/myhost/server1"
  location="myhost"
  locationType="Hostname"
  processId="1095479647062"
  subComponent="Unknown"
  threadId="Alarm : 0"
  componentType="http://www.ibm.com/namespaces/autonomic/WebSphereApplicationServer"/>
```



```

<msgDataElement msgLocale="en_US">
  <msgCatalogTokens value="server1" />
  <msgId>WSVR0024I< /msgId>
  <msgCatalogId>WSVR0024I< /msgCatalogId>
  <msgCatalog>com.ibm.ws.runtime.runtime< /msgCatalog>
</msgDataElement>

<situation categoryName="ReportSituation">
  <situationType xsi:type="ReportSituation" reasoningScope="EXTERNAL" reportCategory="LOG"/>
</situation>

</CommonBaseEvent>

```

A number of extendedDataElement elements in the XML are used by WebSphere Application Server, but are not for application use because these elements might change.

The CommonBaseEvent element defines the Common Base Event instance. This element has a set of attributes that are common for all Common Base Events. This set includes the extensionName attribute, which defines the type or class of the Common Base Event instance, the creation time, severity, and priority.

Nested within the CommonBaseEvent element are elements giving more detail about the situation. The first of these elements is the situation element. This classification is standardized.

The CommonBaseEvent element also includes the sourceComponentId and the (optional) reporterComponentId elements. The sourceComponentId element describes where the situation occurred; the reporterComponentId describes where the situation is detected. If the sourceComponentId and the reporterComponentId elements are the same, the reporterComponentId element is omitted.

The attributes of both the sourceComponentId and the reporterComponentId elements are the same. They identify the component type, name, operating system, and network location. The content of these attributes provides vertical correlation of the stack of IT resources that are active when the Common Base Event is created.

Also included in the CommonBaseEvent element are contextDataElements elements that describe the context in which the situation occurred. This context correlates Common Base Event instances that are part of the same work. This correlation is called *horizontal correlation* because an instance of a particular context type correlates events at the same level of abstraction, for example at the business level, the application level, or at the middleware level.

Extended data elements contain additional data that is used to describe a situation. In this example, an extended data element is added by WebSphere Application Server to describe the Java 2 Platform, Enterprise Edition (J2EE) component that generated the Common Base Event instance and some application data.

Sample Common Base Event template

The content handler uses template information to fill in blanks in the Common Base Event when the Common Base Event complete method is called.

Components that use the WebSphere Application Server event factory home can include a Common Base Event template XML file to provide data to populate Common Base Events. Information that is already supplied in the event is not overridden if the same field is supplied in the template.

The following example illustrates a Common Base Event template:

```

<?xml version="1.0" encoding="UTF-8"?>

<TemplateEvent
  version="1.0.1"

```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="templateEvent.xsd">

<CommonBaseEvent
  <sourceComponentId application="My Application" component="com.ibm.componentX"/>
  <extendedDataElements name="Sample ExtendedDataElement name" type="string">
    <values>Sample ExtendedDataElement value</values>
  </extendedDataElements>
</CommonBaseEvent>

</TemplateEvent>

```

Component identification for problem determination

This topic describes types of problem determination events.

A business application is made up of multiple components. A component can be made up of several internal subcomponents. Consistent application of these concepts is critical for effective problem determination of a business application; all of the parts of the application must use the same concepts and assumptions when creating and formatting events. Use the following definitions and examples when creating Common Base Events for problem determination.

Business application

A business application is the business logic and business data that is used to address a set of specific business requirements. A business application consists of several components of multiple types, combined in a unique manner by an enterprise, to provide the functions and resources that are needed to address those requirements. The primary creator and manager of a business application is the enterprise, and each enterprise or company creates unique business applications. Examples of business applications are the Payroll Application for the ACME Corporation and the Inventory Application for Spacely Sprockets.

Components

A business application is created and managed by the enterprise as a set of components. Components are deployable assets, which are developed either by the enterprise or a vendor, and managed by the enterprise. A component might be created by the enterprise, typically for use within a specific business application. For example, the ACME Corporation might create a set of enterprise beans to represent the business logic that is required by their Payroll Application. A component might also be an asset that is produced by a vendor and acquired by an enterprise. Examples of these components are hardware products, such as IBM eServers or Sun Solaris systems, or software products, such as IBM WebSphere Application Server, Oracle Database Servers.

Subcomponents

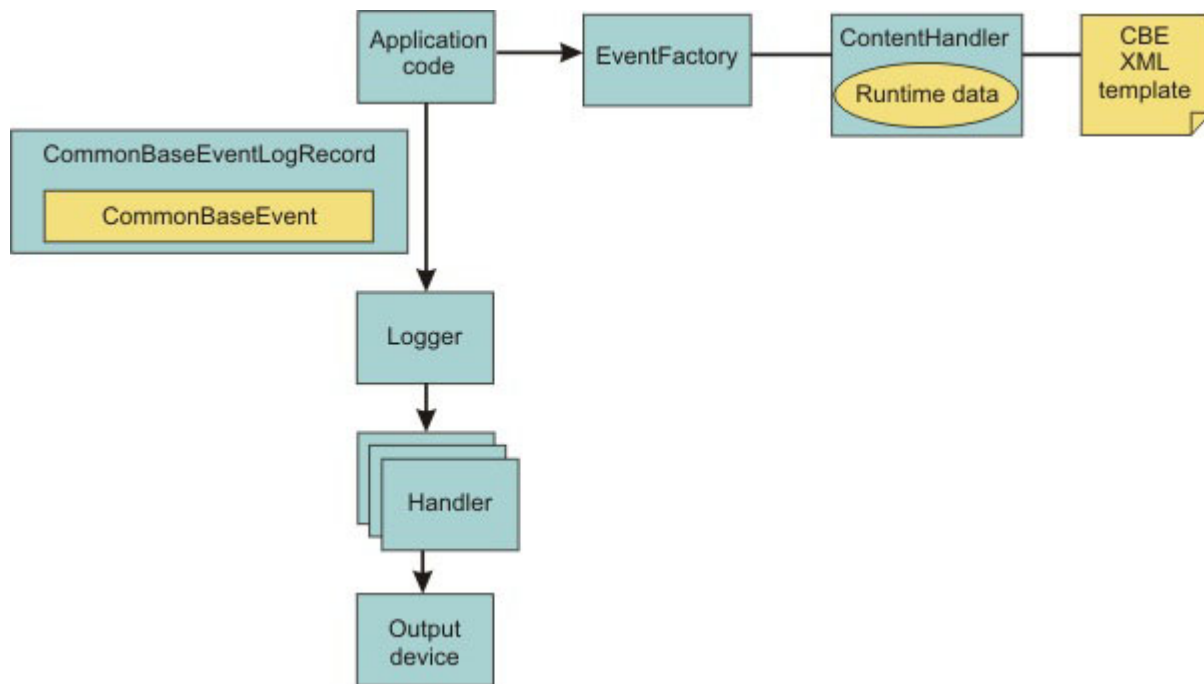
A specific component, depending on its complexity, might consist of several subcomponents. For example, the IBM WebSphere Application Server consists of many subcomponents, such as the enterprise bean container and the servlet engine. Subcomponent information is typically used only by the creator of the component to service the component, and as such are not separately deployable or manageable resources in the enterprise. The enterprise might deploy a change or update to a subcomponent, but only upon guidance from the component vendor and as part of the vendor's component. For example, a software fix for the enterprise bean container of the IBM WebSphere Application Server is packaged and deployed as a software update to the IBM WebSphere Application Server. Replacement of the processor in an IBM eServer is deployed as a physical part, but only as a part of the original deployed component, the IBM eServer.

Logging with Common Base Event API and the Java logging API

In cases where the events that are generated by the Java logging API are insufficient to describe the event that needs capturing, you can create Common Base Events with the Common Base Event factory APIs.

Before you begin

When you create a Common Base Event, you can add data to the Common Base Event before it is logged. The following diagram illustrates how application code can create and log Common Base Events:



About this task

WebSphere Application Server is configured to use an event factory that automatically populates WebSphere Application Server-specific information into the Common Base Events that it generates. In general, it is good practice to create events using the WebSphere Application Server default Common Base Event factory because this approach ensures consistency of Common Base Event content across events. However, you can create and use other Common Base Event factories.

Common Base Events are initiated and logged in the following sequence:

1. Application code invokes the `createCommonBaseEvent` method on the `EventFactory` class to create a `CommonBaseEvent`.
2. Application code wraps `CommonBaseEvent` event in a `CommonBaseEventLogRecord` record, and adds event-specific data.
3. Application code calls the `CommonBaseEvent` event `complete` method.
4. The `CommonBaseEvent` event invokes the `ContentHandler` `completeEvent` method.
5. The `ContentHandler` handler adds XML template data to the `CommonBaseEvent` event. Not all `ContentHandler` handlers support templates.
6. The `ContentHandler` handler adds runtime data to the `CommonBaseEvent` event.
7. Application code passes the `CommonBaseEventLogRecord` record to the logger using the `Logger.log` method.
8. Logger passes `CommonBaseEventLogRecord` record to Handlers.
9. Handlers format data and write to the output device.

Procedure

- You can use the default Common Base Event factory to generate content. Read “Generate Common Base Event content with the default event factory” on page 90 for more information.

- If you do not wish to use the default event factory, you can create custom content handlers and event factories.
 1. Create a custom factory home. Read “Creating custom Common Base Event factory homes” on page 94.
 2. Create a custom content handler. Read “Creating custom Common Base Event content handlers” on page 93.

Results

After completing all the above steps you will have a Common Base event based on your configuration settings.

Generate Common Base Event content with the default event factory

A default Common Base Event content handler populates Common Base Events with WebSphere Application Server runtime information. This content handler can also use a Common Base Event template to populate Common Base Events.

The default content handler is used when the server creates `CommonBaseEventLogRecords` as would be the case in the following example:

```
// Get a named logger
Logger logger = Logger.getLogger("com.ibm.someLogger");
// Log to the logger -- implicitly the default content handler
// will be associated with the CommonBaseEvent contained in the
// CommonBaseEventLogRecord.
logger.warning("MSG_KEY_001");
```

To specify a Common Base Event template in the previous case, a `Logger.properties` file would need to be provided with an `eventfactory` entry for `com.ibm.someLogger`. If a valid template is found on the classpath, then the `Logger`'s event factory will use the specified template's content in addition to the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the `Logger`'s event factory will only use the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler is also associated with the event factory home supplied in the global event factory context. This is convenient for creating Common Base Events that need to be populated with content similar to that generated from the WebSphere Application Server:

```
// Request the event factory from the global event factory home
EventFactory eventFactory =
    EventFactoryContext.getInstance().getEventFactoryHome().getEventFactory(templateName);

// Create a Common Base Event
CommonBaseEvent commonBaseEvent = eventFactory.createCommonBaseEvent();

// Complete the Common Base Event using content from the template (if specified previously)
// and the server runtime information.
eventFactory.getContentHandler().completeEvent(commonBaseEvent);
```

In the previous example, if the template referenced by `templateName` is found on the classpath, and the template is valid, then the event factory home will return an event factory which uses a content handler that combines the template's content with the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the event factory home will return an event factory which uses a content handler that uses only the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler populates Common Base Events in the server environment with the following runtime information:

CommonBaseEvent.globalInstanceId

Value: The unique_record_id

Set this value only if the CommonBaseEvent.globalInstanceId value is null before the completeEvent method is called.

CommonBaseEvent.msg

Value: A localized message that is based on the MsgDataElement element.

Set this value only if the CommonBaseEvent.msg message is null before the completeEvent method is called.

CommonBaseEvent.severity

Value: Set based on the value of level set on the CommonBaseEventLogRecord record, if level >= Level.SEVERE, set to 50; if level >= Level.WARNING, set to 30; the default is set to 10.

Set this value only if the CommonBaseEvent.severity value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.component

Value: Set based on the LoggerName value that is set on the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.component is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.componentIdType

Value: "Unknown"

Set this value only if the CommonBaseEvent.ComponentIdentification.componentIdType value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.executionEnvironment

Value: OSname[OSarch]#OSversion

Set this value only if the CommonBaseEvent.ComponentIdentification.executionEnvironment value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.instanceId

Value: cellName\nnodeName\nserverName

Set this value only if the CommonBaseEvent.ComponentIdentification.instanceId value is null before the completeEvent method is called. Set only in a server environment because this value is ignored in a client application.

CommonBaseEvent.ComponentIdentification.location

Value: The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.locationType

Value: The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.processId

Value: An internally generated representation of the process number.

Set this value only if the CommonBaseEvent.ComponentIdentification.processId value is null before the completeEvent method is called

CommonBaseEvent.ComponentIdentification.subComponent

Value: Set based on values of the sourceClassName and the sourceMethodName names that are set on the sourceClassName.sourceMethodName name of the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.subComponent values is null before the completeEvent method is called and both the sourceClassName and the sourceMethodName names are set.

CommonBaseEvent.ComponentIdentification.threadId

Value: Set to the value of the Java Virtual Machine (JVM) thread name.

Set this value only if the CommonBaseEvent.ComponentIdentification.threadId values is null before the completeEvent value is called.

CommonBaseEvent.ComponentIdentification.componentType

Value: <http://www.ibm.com/namespaces/autonomic/WebSphereApplicationServer>

Set this value only if the CommonBaseEvent.ComponentIdentification.componentType values is null before the completeEvent method is called.

CommonBaseEvent.MsgDataElement.msgLocale

Value: Set based on the default locale of the JVM.

Set this value only if the CommonBaseEvent.msg value is null before the completeEvent method is called.

CommonBaseEvent.Situation.categoryName

Value: ReportSituation

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.type

Value: ReportSituation

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reasoningScope

Value: EXTERNAL

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reportCategory

Value: LOG

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

The sourceComponentIdentification value is populated if no reporterComponentIdentification ID exists when the completeEvent method is invoked on the content handler. Otherwise, the reporterComponentIdentification ID is populated instead.

Common Base Event content handler

Content handlers populate data into Common Base Events when the Common Base Event complete method is invoked. You can associate content handlers with Common Base Event templates, which provide default information to transfer into each Common Base Event.

Content handlers might also provide any other information that is relevant to completing the population of the Common Base Event, such as appropriate runtime defaults. The use of content handlers ensures consistency of field use in the Common Base Event within a component or within a set of components that share the same runtime. For example, some content handlers support the specification of a template. If

used consistently across a component, this template ensures that all events for that component have the same template information filled in. Similarly, some content handlers can also supply runtime information to their associated Common Base Events. If consistently used throughout the entire runtime, runtime information ensures that all events use runtime data in a similar way.

The event factory home that is used in the WebSphere Application Server runtime is associated with a content handler that both reads from a template, and supplies runtime data. Have components use Event Factories that are obtained from this event factory home with their own templates, to produce consistency between application events and server events.

More details can be found in “Creating custom Common Base Event content handlers” or the API documentation for `org.eclipse.hyades.logging.events.cbe.ContentHandler` at <http://www.eclipse.org/tptp/index.html>.

Creating custom Common Base Event content handlers

Create a custom Common Base Event content handler or template to automate configuration or values for specific events.

Before you begin

A *content handler* is an object that automatically sets the property values of each event based on any arbitrary policies that you want to use.

The following content handler classes were added to WebSphere Application Server to facilitate the use of the Common Base Event infrastructure:

Class name	Description
<code>WsContentHandlerImpl</code>	This provides an implementation of <code>org.eclipse.hyades.logging.events.cbe.ContentHandler</code> specifically for use in the WebSphere Application Server environment. This content handler completes Common Base Events using information from the WebSphere Application Server runtime, and it uses the same content handler as is used internally by the WebSphere Application Server when completing Common Base Events for logging.
<code>WsTemplateContentHandlerImpl</code>	This provides the same function as <code>WsContentHandlerImpl</code> , but it extends the <code>org.eclipse.hyades.logging.events.cbe.impl.TemplateContentHandlerImpl</code> class to enable the use of a Common Base Event template. Template content takes precedence in cases where the template data specifies values for the same Common Base Event fields as does the <code>WsContentHandlerImpl</code> .

About this task

In some situations, you might want some event property data set automatically for every event that you create. This automation is a way to fill in certain standard values that do not change, such as the application name, or to set some properties based on information that is available from the runtime environment, like creation time or thread information. You can set property data automatically by creating a content handler.

Procedure

- Use the following code sample to implement the `CustomContentHandler` class:

```
public class CustomContentHandler extends WsContentHandlerImpl {

    public CustomContentHandler() {
        super();
        // TODO Custom initialization code goes here
    }
}
```



```

}

public void completeEvent(CommonBaseEvent cbe) throws CompletionException {
    // following code will add WAS content to the Content Base Event
    super.completeEvent(cbe);
    // TODO Custom content can be added to the Content Base Event here
}
}

```

- The following shows how to implement the CustomTemplateContentHandler class:

```

public class CustomTemplateContentHandler extends WsTemplateContentHandlerImpl {

    public CustomTemplateContentHandler() {
        super();
        // TODO Custom initialization code goes here
    }

    public void completeEvent(CommonBaseEvent cbe) throws CompletionException {
        // following code will add WAS content to the Content Base Event
        super.completeEvent(cbe);
        // TODO Custom content can be added to the Content Base Event here
    }
}

```

Results

You now have a content handler or a custom content handler template based on the settings that you specified.

Common Base Event factory home

Event Factory homes provide Event Factory instantiation that is based on a unique factory name.

Event factory home implementations are tightly coupled with content handlers that are used to populate Common Base Events with template or default data. Event factory instances are maintained by the associated event factory home, based on their unique name. For example, when application code requests a named event factory, the newly created Event Factory instance is returned and persisted for future requests for that named event factory. An abstract event factory home class provides the implementation for the APIs in the event factory home interface. Implementers extend the abstract event factory home class and implement the createContentHandler API to create a typed content handler that is based on the type of event factory home implementation.

In WebSphere Application Server, the default event factory home that is obtained with a call to EventFactoryContext.getInstance.getEventFactoryHome method is associated with a ContentHandler handler capable of supplying both event template information, as well as WebSphere Application Server runtime default information.

More details can be found in the API documentation for `org.eclipse.hyades.logging.events.cbe.EventFactoryHome` at www.eclipse.org/hyades.

Creating custom Common Base Event factory homes

Use custom Common Base Event factory homes to control configuration and implementation of unique event factories.

Before you begin

Event factory homes create and provide homes for Event Factory instances. Each event factory home has a content handler. This content handler is assigned to every event factory the event factory home creates. In turn, when a Common Base Event is created, the content handler from the event factory is assigned to it. Event factory instances are maintained by the associated event factory home, based on their unique

name. For example, when application code requests a named event factory, the newly created event factory instance is returned and persisted for future requests for that named event factory.

The following classes were added to facilitate the use of event eactory homes for logging Common Base Events:

Class name	Description
WsEventFactoryHomeImpl	This class extends the org.eclipse.hyades.logging.events.cbe.impl.AbstractEventFactoryHome class. This event factory home returns event factory instances associated with the WsContentHandlerImpl content handler. The WsContentHandlerImpl is the content handler used by the WebSphere Application Server by default when no event factory template is in use.
WsTemplateEventFactoryHomeImpl	This class extends the org.eclipse.hyades.logging.events.cbe.impl.EventXMLFileEventFactoryHomeImpl class. This event factory home returns event factory instances associated with the WsTemplateContentHandlerImpl Content Handler. The WsTemplateContentHandlerImpl is the content handler used by the WebSphere Application Server when an Event Factory template is required.

About this task

Custom event factory homes support the use of Common Base Event for logging in WebSphere Application Server and make logging easy and consistent between the WebSphere Application Server runtime and the exploiters of this API. The CustomEventFactoryHome and CustomTemplateEventFactoryHome classes will be used to obtain an event factory. These classes are there to make sure the correct content handler is being used with a particular event factory. The CustomEventFactoryHelper class is an example of how the infrastructure provider can hide the factory selection details from infrastructure users, using their own set of parameters to decide which the appropriate event factory is.

Procedure

- The following code samples provide examples of how to implement and use the CustomEventFactoryHome class.

- Implementation of the CustomEventFactoryHome class is as follows:

```
public class CustomEventFactoryHome extends AbstractEventFactoryHome {

    public CustomEventFactoryHome() {
        super();
        // TODO Custom intialization code goes here
    }

    public ContentHandler createContentHandler(String arg0) {
        // Always use custom content handler
        return resolveContentHandler();
    }

    public ContentHandler resolveContentHandler() {
        // Always use custom content handler
        return new CustomContentHandler();
    }
}
```

- The following is an example of how to use the CustomEventFactoryHome class:

```
// get the event factory
EventFactory eventFactory=(new CustomEventFactoryHome()).getEventFactory("XYZ");
// create an event - call appropriate method
eventFactory.createCommonBaseEvent();
// log event ...
```

- For the CustomTemplateEventFactoryHome class you can use the following code for implementation and use:

1. Implement the CustomTemplateEventFactoryHome class by using this code:

```
public class CustomTemplateEventFactoryHome extends
    EventXMLFileEventFactoryHomeImpl {

    public CustomTemplateEventFactoryHome() {
        super();
        // TODO Custom initialization code goes here
    }

    public ContentHandler createContentHandler(String arg0) {
        // Always use custom content handler
        return resolveContentHandler();
    }

    public ContentHandler resolveContentHandler() {
        // Always use custom content handler
        return new CustomTemplateContentHandler();
    }
}
```

2. Use the CustomTemplateEventFactoryHome class by following this sample code:

```
// get the event factory
EventFactory eventFactory=(new
    CustomTemplateEventFactoryHome()).getEventFactory("XYZ");
// create an event - call appropriate method
eventFactory.createCommonBaseEvent();
// log event ...
```

- The CustomEventFactoryHelper class can be implemented and used by following the code below:

1. Implement the custom CustomEventFactoryHelper class using this code:

```
public class CustomTemplateEventFactoryHome extends
    EventXMLFileEventFactoryHomeImpl {

    public CustomTemplateEventFactoryHome() {
        super();
        // TODO Custom initialization code goes here
    }

    public ContentHandler createContentHandler(String arg0) {
        // Always use custom content handler
        return resolveContentHandler();
    }

    public ContentHandler resolveContentHandler() {
        // Always use custom content handler
        return new CustomTemplateContentHandler();
    }
}
```

Figure 4 CustomTemplateEventFactoryHome class

```
public class CustomEventFactoryHelper {
    // name of the event factory to use
    public static final String FACTORY_NAME="XYZ";

    public static EventFactory getEventFactory(String param1, String param2) {
        EventFactory factory=null;
        switch (resolveFactory(param1,param2)) {
            case 1:
                factory=(new CustomEventFactoryHome()).getEventFactory(FACTORY_NAME);
                break;
            case 2:
                factory=(new
                    CustomTemplateEventFactoryHome()).getEventFactory(FACTORY_NAME);
                break;
        }
    }
}
```

```

default:
    // Add default for event factory
    break;
}
return factory;
}

private static int resolveFactory(String param1, String param2) {
    int factory=0;
    // Add code here to resolve which factory to use
    return factory;
}
}

```

2. To use the CustomEventFactoryHelper class, use the following code:

```

// get the event factory
EventFactory eventFactory=
    CustomEventFactoryHelper.getEventFactory("param1","param2","param3");
// create an event - call appropriate method
eventFactory.createCommonBaseEvent();
// log event ...

```

Results

Use the information provided here to implement a custom content factory home and the associated classes based on the settings that you specify.

Common Base Event factory context

The event factory context provides a service to look up event factory homes. Retrieve the event factory context using a call to the EventFactoryContext.getInstance method.

Using this class, you can look up the event factory homes by name, and avoid the need to include the typed home in code. The EventFactoryHome name must be located on the class path to be found. The EventFactoryContext context also stores an EventFactoryHome name as a default, which can be obtained with a call to the EventFactoryContext.getInstance.getEventFactoryHome method.

In WebSphere Application Server, the EventFactoryContext context is configured with a default EventFactoryHome name which is associated to a ContentHandler handler that is capable of supplying both event template information, as well as WebSphere Application Server runtime default information.

More details can be found in the API documentation for `org.eclipse.hyades.logging.events.cbe.EventFactory` at www.eclipse.org/hyades.

Common Base Event factory

Use event factories to create Common Base Events and complete event properties with associated content handlers.

Content handlers populate data into Common Base Events when the Common Base Event invokes the complete method. All event properties set by the application code have priority over all properties that are specified by the content handler. Event factory implementations are tightly coupled with the content handler instance, which is associated with the event factory when the event factory is instantiated. Factory instances can be retrieved only from their associated event factory home. Event factory instances are retrieved and maintained based on unique names. Event factory names are hierarchical; they are represented using the standard Java dot-delimited, name-space naming conventions.

More details can be found in the API documentation for `org.eclipse.hyades.logging.events.cbe.EventFactory` at www.eclipse.org/hyades.

java.util.logging -- Java logging programming interface

The `java.util.logging.Logger` class provides a variety of methods with which data can be logged.

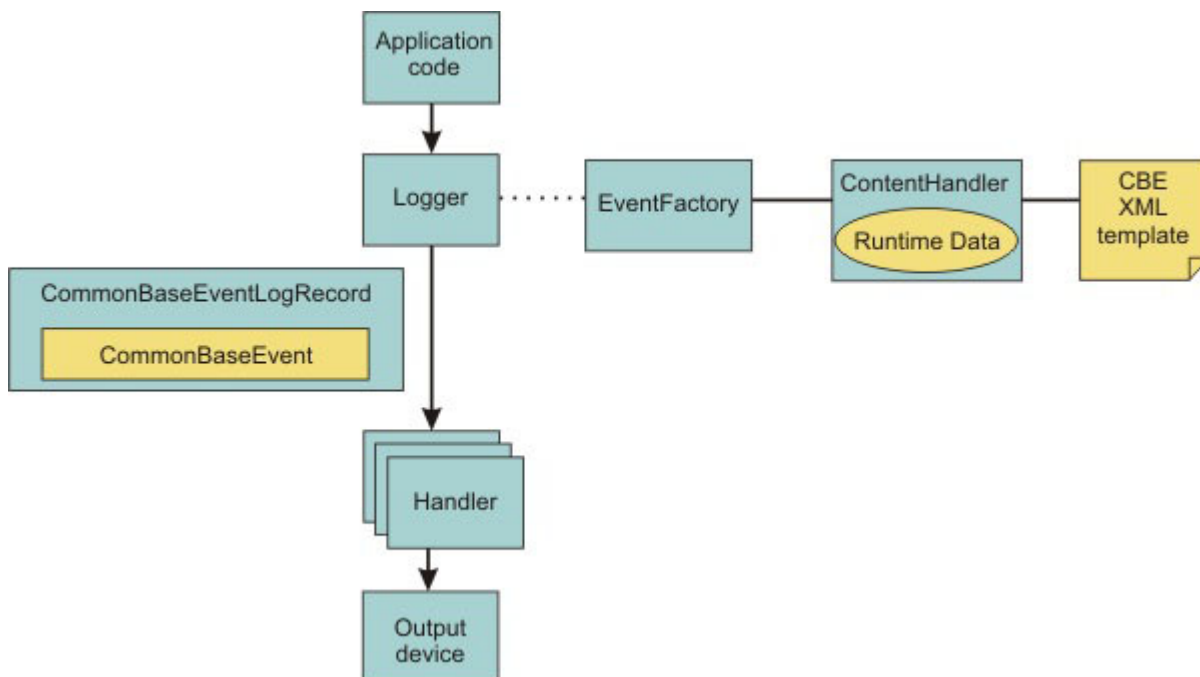
In the WebSphere Application Server, when using basic log and trace mode, the Java logging API (`java.util.logging`) automatically creates Common Base Events for events that are logged at the `WsLevel.DETAIL` level or later (including `WsLevel.DETAIL`, `Level.CONFIG`, `Level.INFO`, `WsLevel.AUDIT`, `Level.WARNING`, `Level.SEVERE`, and `WsLevel.FATAL`). These Common Base Events are created using the event factory that is associated with the logger to which the message is logged. If no event factory is specified, WebSphere Application Server uses a default event factory which automatically fills in WebSphere Application Server-specific information.

The WebSphere Application Server uses a special implementation of the `java.util.logging.Logger` class that automatically creates Common Base Events for the following methods:

- `config`
- `info`
- `warning`
- `severe`
- `log`: All variants except `log(LogRecord)` when used with the `WsLevel.DETAIL` level or more severe levels
- `logp`: When used with the `WsLevel.DETAIL` level or more severe levels
- `logrb`: When used with the `WsLevel.DETAIL` level or more severe levels

The WebSphere Application Server logger implementation is used only for named loggers for example, loggers that are instantiated with calls, such as `Logger.getLogger("com.xyz.SomeLoggerName")`. Loggers instantiated with calls to the `Logger.getAnonymousLogger` and `Logger.getLogger`, or `Logger.global` methods do not use the WebSphere Application Server implementation, and do not automatically create Common Base Events for logging requests made to them. Log records that are logged directly with the `Logger.log(LogRecord)` method are not automatically converted by WebSphere Application Server loggers into Common Base Events.

The following diagram illustrates how application code can log Common Base Events:



The Java logging API processing of named loggers and message-level events proceeds as follows:

1. Application code invokes the named logger (WsLevel.DETAIL or later) with event-specific data.
2. The logger creates a Common Base Event using the createCommonBaseEvent method on the event factory that is associated with the logger.
3. The logger creates a Common Base Event using the event factory associated to the logger.
4. The logger wraps the common base event in a CommonBaseEventLogRecord record, and adds event-specific data.
5. The logger calls the Common Base Event complete method.
6. The Common Base Event invokes the ContentHandler completeEvent method.
7. The content handler adds XML template data to the Common Base Event (including for example, the component name). Not all content handlers support templates.
8. The content handler adds runtime data to the Common Base Event (including for example, the current thread name).
9. The logger passes the CommonBaseEventLogRecord record to the handlers.
10. The handlers format data and write to the output device.

Logger.properties file

Use the `Logger.properties` file to set logger attributes for your component.

The properties file is loaded the first time the `Logger.getLogger(loggername)` method is called within an application. The `Logger.properties` file must be either on the WebSphere Application Server class path, or the context class path.

The logging subsystem uses Common Base Events to represent all the messages in the WebSphere Application Server `activity.log` file. You can specify your own event factory template to be used with your loggers. Use the `eventfactory` property in your `Logger.properties` file. See “Sample Common Base Event template” on page 87 for details on the Common Base Event template.

By convention, the name of the event factory template file should be the fully qualified package name of the package using the template. The name of the file must end with the `.event.xml` extension. For example, a valid event factory template file name for the `com.abc.somepackage` package is:

```
com.abc.somepackage.event.xml
```

When you specify the property value for the `eventfactory` property in the `Logger.properties` file, include the full path name with no leading slash relative to the root of your class path entry. Do not include the `.event.xml` extension.

For example, if the template files from the previous example are located in the `com/abc/templates` directory, the valid value for the `eventfactory` property is:

```
com/abc/templates/com.abc.somepackage
```

Finally, if this event factory template file is used by the `com.abc.somepackage.SomeClass` logger, then the following entry will appear in the `Logger.properties` file:

```
com.abc.somepackage.SomeClass.eventfactory=com/abc/templates/com.abc.somepackage
```

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Logging Common Base Events in WebSphere Application Server

The following practices ensure consistent use of Common Base Events within your components, and between your components and WebSphere Application Server components.

Follow these guidelines:

- Use a different logger for each component. Sharing loggers across components gets in the way of associating loggers with component-specific information.
- Associate loggers with event templates that specify source component identification. This association ensures that the source of all events created with the logger is properly identified.
- Use the same template for directly created Common Base Events (events created using the Common Base Event factories) and indirectly created Common Base Events (events created using the Java logging API) within the same component.
- Avoid calling the complete method on Common Base Events until you are finished adding data to the Common Base Event and are ready to log it. This approach ensures that any decisions made by the content handler based on data already in the event are made using the final data.

The following sample `Logger.properties` file entry demonstrates how to associate the `com.ibm.componentX` logger with the `com.ibm.componentX` event factory:

```
com.ibm.componentX.eventfactory=com.ibm.componentX
```

The following sample code demonstrates the use of the same event factory setting for direct (Part 1) and indirect (Part 2) Common Base Event logging:

```
<?xml version="1.0" encoding="UTF-8"?>

<TemplateEvent>
  version="1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="templateEvent.xsd">

  <CommonBaseEvent>
    <sourceComponentId application="My application" component="com.ibm.componentX"/>
    <extendedDataElements CommonBaseEventname="Sample ExtendedDataElement name" type="string">
      <values>Sample ExtendedDataElement value</values>
    </extendedDataElements>
  </CommonBaseEvent>

</TemplateEvent>
```

Showlog commands for Common Base Events


The `showlog` command converts the service log from binary format into plain text.

Purpose

These `showlog` commands to produce output in Common Base Event XML format.

-  `showlog -format CBE-XML-1.0.1 filename`

where:

 `filename`
Is the service log file name.

For examples of `showlog` scripts, see [Viewing the service log](#).

Chapter 9. Overview and new features for deploying applications

View the topics in the following list to learn more about installing applications or modules on product deployment targets.

What is new for deployers

This topic provides an overview of new and changed features of the programming model and application serving environment as it pertains to deployment.

Learn about WebSphere applications: Overview and new features

This topic provides an overview of the programming model.

Ways to install enterprise applications or modules

This topic provides lists the ways to install Java Platform, Enterprise Edition (Java EE) application files on product deployment targets.

Accessing the samples

The samples are a good way to become familiar with the programming model.

Chapter 10. Deploying applications to the Liberty profile

You can deploy web applications or OSGi applications to the Liberty profile. You deploy an application by either dropping the application into a previously-defined “dropins” directory, or by adding an application entry to the server configuration.

Before you begin

This topic assumes that you have not disabled dynamic updates to the runtime configuration, as described in Controlling dynamic updates.

About this task

By default, the “dropins” directory is automatically monitored. If you drop an application into this directory, the application is automatically deployed on the server. Similarly, if the application is deleted from the directory, the application is automatically removed from the server. The “dropins” directory can be used for applications that do not require additional configuration, such as security role mapping. You do not have to include the application entry or any relevant information in the server configuration. For applications that are not in the “dropins” directory, you specify the location using an application entry in the server configuration. The location can be on the file system, or at a URL.

Your application can be packaged as an archive file or as a directory. For applications in the “dropins” directory, the file name and file extension are used by the application monitor to determine the type of application, and to generate the application name and (for web applications) the context-root. For example, if the archive file or directory is named `snoop.war`, the application monitor assumes that the application is a web application, that the application name is “snoop”, and that the context-root is also snoop. For configured applications, you specify the application type and name, and (for web applications) the application name is also used as the context-root.

For more information about the default directory structure, and the properties that are associated with directories (for example `server.config.dir`), see Liberty profile: Directory locations and properties.

Procedure

- Deploy an application by dropping it into the “dropins” directory.

For example, using the default directory structure, to deploy an application you drop it into the `${server.config.dir}/dropins` directory (that is, `wlp/usr/servers/server_name/dropins`).

- Deploy an application by adding it to the server configuration.

Configure the **application** element in the `server.xml` configuration file. See Liberty profile: Configuration elements in the `server.xml` file. You must configure the following attributes for the application:

- **id** Must be unique and is used internally by the server.
- **name** Must be unique and depending on the application. It might be used as the context-root of the application. For more information on how the context-root is set for an application, see “Deploying a web application to the Liberty profile” on page 115.
- **type** Specifies the type of the application. The supported application types are **war**, **ear** and **eba**.
- **location** Specifies the location of the application. It can be an absolute path or a URL which you can download the application from. It can also be the file name of your application (including file extension if any).

If the application is available on the file system, the location can either be the full path name or a simple file name. If the location does not include the full path, the application manager looks for the application

in `${server.config.dir}/apps` and `${shared.app.dir}`. If the application is available at a URL, the application manager downloads the application to a temporary folder inside the server work area, then starts the application.

Note: The location that you specify for a configured application should not be in the “dropins” directory. If you drop an application into the “dropins” directory, and also specify the location in the `server.xml` file, you are telling the server to deploy the application twice.

In the following two examples, the location is the file system. If the location is a URL, enter the URL in the location field.

```
<application id="ImpactEBA" name="ImpactEBA" type="eba" location="D:/apps/ImpactEBA.eba"/>
<application id="ImpactWeb" name="ImpactWeb" type="war" location="ImpactWeb.war"/>
```

The second example does not include the full path. In this case, you must put the application in one of the following locations:

- `${server.config.dir}/apps` (that is, *server_directory/user/servers/server_name/apps*)
- `${shared.app.dir}` (that is, *liberty_install_location/usr/shared/apps*)

Notes:

- You must create the server-level apps directory, whereas the shared apps directory is present by default. See Liberty profile: Directory locations and properties for more information about the properties associated with the server directories.
 - The **application** element can be set before or after the server has started. If the element is set after the server has started, the changes are picked up dynamically.
- Remove an application.

For applications that are included in the server configuration, remove the reference to the application from the `server.xml` file. The application is then automatically removed from the server.

For applications that are deployed to the “dropins” directory, delete the application from the directory. The application is then automatically removed from the server.

To uninstall all applications that are in the “dropins” directory, set the application monitor `dropinsEnabled` property to `false` as described in Controlling dynamic updates.

What to do next

For all deployed applications, you can configure if application monitoring is enabled, and how often to check for updates to applications. For the “dropins” directory, you can also configure the name and location of the directory, and choose whether or not to deploy the applications that are in the directory. See Controlling dynamic updates.

Packaging a Liberty profile server from the command prompt

From the command prompt you can create a compressed file containing a server runtime environment, server configuration, and applications.

About this task

Because a Liberty server is lightweight, you might find it useful to package up your applications and server in a compressed file. You can then store this package, distribute it to colleagues, use it to deploy the application to a different location or to another machine, or even embed it in your product distribution.

Procedure

1. Open a command prompt, then change directory to the `wlp` directory.
2. Stop the server.
3. Package the server.

Run the following command. If you do not specify a server name, `defaultServer` is used. If you do not specify the `--archive` parameter, the value of `server_name` is used for `package_file_name`, and the compressed file is created in the `${server.output.dir}` directory.

You can also use the `--include` option with this command. For example, `--include=all` option packages all the files under the server installation directory; `--include=use` option packages files in the `${WLP_USER_DIR}` directory.

Results

If the specified server does not exist, the command does not succeed. If the specified server exists, a compressed file is created that contains your applications and server.

Using JNDI binding for constants from the server configuration files

You can bind constants into the default JNDI namespace from the server configuration files using the `jndiEntry` element on the Liberty profile.

About this task

The default JNDI namespace is available in the Liberty profile to provide bindings to miscellaneous objects needed by applications. Any data sources declared in the server configuration files are available in the default JNDI namespace. Additionally, you can bind Java strings and primitive data types in the configuration file into JNDI namespace. These constants are then made available to an application at run time, providing a very simple and portable way to pass configuration values into the application.

See Naming for more information about JNDI.

Following steps show how to bind the constants and use them in your application.

Procedure

1. To add a constant into the default JNDI namespace, the `jndi-1.0` server feature must be specified in the `server.xml` file of the Liberty profile server.

```
<featureManager>
  <feature>jndi-1.0</feature>
</featureManager>
```

2. To bind constants into the JNDI namespace, add `jndiEntry` elements into the `server.xml` file by specifying the `jndiName` and `value` attributes.

```
<jndiEntry jndiName="schoolOfAthens/defaultAdminUserName" value="plato" />
<jndiEntry jndiName="schoolOfAthens/defaultAdminPassword" value="republic" />
```

3. To look up the constants from an application using a JNDI context, use the following code:

```
Object jndiConstant = new InitialContext().lookup("schoolOfAthens/defaultAdminUserName");
String defaultAdmin = (String) jndiConstant;
```

Note:

- The `lookup()` method returns an object to the application. The type of the object is determined by interpreting the value stored in the `jndiEntry` element as a Java literal string or primitive data type. If the parsing fails, the exact value is provided as an unmodified string.
- The `jndiEntry` element supports the integer, floating-point, boolean, character and string literals as described in the Java Language Specification, Java SE 7 Edition, section 3.10. String and character literals might contain unicode escaped sequences (section 3.3), and the octal and character escape sequences (section 3.10.6). The null literal (section 3.10.7) and class literals (section 15.8.2) are not supported.

See the following examples of Java literals:

- The string "Hello, world" followed by a newline:
`<jndiEntry jndiName="a" value="Hello, world.\n" />`
- The integer with a binary value 1010101:
`<jndiEntry jndiName="b" value="0b1010101" />`
- The single character 'X':
`<jndiEntry jndiName="c" value="'X'" />`
- The double-precision floating point number 1.0:
`<jndiEntry jndiName="d" value="1.00" />`

See Liberty profile: Configuration elements in the `server.xml` file for more information about `jndiEntry` element.

Sharing common OSGi bundles for the Liberty profile

You can share common OSGi bundles by placing them in a directory and configure the `server.xml` file for your server so that those common OSGi bundles are available to your enterprise applications.

Procedure

- Create a directory in your file system and place all the common OSGi bundles into the directory.
- Add the following lines into the `server.xml` file.

```
<bundleRepository>
  <fileset dir="directory_path" include="*.jar"/>
</bundleRepository>
```

Where `directory_path` is the path to the directory that contains the common OSGi bundles.

- Define a dependency on the common bundle using `import` phrase in the `manifest.mf` file of your application.

Configuring class loaders for Java EE applications

By default, each application can access a set of provided APIs, as well as being able to access its own internal classes and libraries. You can override the default settings, and configure class loading for each application.

About this task

Each Java EE application has its own class loader in a running Liberty profile server. The Liberty profile assumes some default settings for all Java EE applications, so that they can access the supported specification APIs (for example the Servlet APIs if the Servlet feature is enabled), and the IBM APIs. By default, each application can access these provided APIs, as well as being able to access its own internal classes and libraries. If you need to override the default settings and configure class loading for your application, complete one or more of the following tasks.

Note: If you use configuration to override the default settings, you cannot deploy the application by dropping it into the “dropins” directory.

Procedure

- “Using a Java library with a Java EE application” on page 107
- “Sharing a library across multiple Java EE applications” on page 107
- “Accessing third-party APIs from a Java EE application” on page 108
- “Removing access to third-party APIs for a Java EE application” on page 109
- “Overriding a provided API with an alternative version” on page 109
- “Providing global libraries for all Java EE applications” on page 110

Using a Java library with a Java EE application

One way of using Java libraries with an application is to include them in the application itself. This might not always be desirable or appropriate, especially if the application is already packaged and does not include the library.

About this task

In the following example, a library called Alexandria consists of two files:

- alexandria-scrolls.jar and
- commons-lang.jar

An application called Scholar, running on a server called Academy, needs access to this library.

Procedure

1. Create a lib/Alexandria directory in the servers/Academy directory under the `${WLP_USER_DIR}` directory.

For example: `wlp/usr/servers/Academy/lib/Alexandria`.

2. Copy the alexandria-scrolls.jar and commons-lang.jar files into the new folder.
3. Configure class loading for the application, so that the Alexandria library is loaded.

In the server.xml file, or an included file, add the following code:

```
<application id="scholar" name="Scholar" type="ear" location="scholar.ear">
  <classloader>
    <privateLibrary>
      <fileset dir="${server.config.dir}/lib/Alexandria" includes="*.jar" scanInterval="5s" />
    </privateLibrary>
  </classloader>
</application>
```

Note: The `<privateLibrary>` element can also take a `filesetRef` attribute with a comma-separated list of `<fileset>` element IDs.

Sharing a library across multiple Java EE applications

Libraries can be shared across multiple Java EE applications. All the applications can use the same classes at run time, or each application can use its own separate copy of those classes loaded from the same location.

About this task

In the following example, a library called Alexandria consists of two files:

- alexandria-scrolls.jar and
- commons-lang.jar

An application called Scholar and an application called Student are running on a server called Academy, and both need access to this library.

Procedure

1. Create a lib/Alexandria directory in the servers/Academy directory under the `${WLP_USER_DIR}` directory.

For example: `wlp/usr/servers/Academy/lib/Alexandria`.

2. Copy the alexandria-scrolls.jar and commons-lang.jar files into the new folder.
3. Configure class loading for the application, so that the Alexandria library is loaded.

In the server.xml file, or an included file, define the library by adding the following code:

```
<library id="Alexandria">
  <fileset dir="${server.config.dir}/lib/Alexandria" includes="*.jar" scanInterval="5s" />
</library>
```

Note: The `<library>` element can also take a `filesetRef` attribute with a comma-separated list of `<fileset>` element IDs.

4. Reference the library from the applications, so that both these applications share a single copy of the library.

In the `server.xml` file, or an included file, add the following code:

```
<application id="scholar" name="Scholar" type="ear" location="scholar.ear">
  <classloader commonLibraryRef="Alexandria" />
</application>
```

```
<application id="student" name="Student" type="ear" location="student.ear">
  <classloader commonLibraryRef="Alexandria" />
</application>
```

Note: The `<commonLibraryRef>` element can take a comma-separated list of library IDs.

5. Optional: Configure another application to have its own set of classes loaded from the same JAR files.

If another application called (for example) `Spy` needs its own copy of the classes, the same physical files on disk can be used. In the `server.xml` file, or an included file, add the following code:

```
<application id="spy" name="Spy" type="war" location="spy.war">
  <classloader privateLibraryRef="Alexandria" />
</application>
```

Note: The `<privateLibraryRef>` element can take a comma-separated list of library IDs.

Accessing third-party APIs from a Java EE application

By default, Java EE applications do not have access to the third-party APIs available in the Liberty profile. To enable this access, the application must be configured in the `server.xml` file, or an included file.

About this task

In the following example, an application called `Scholar` needs access to the third-party APIs that are available in the Liberty profile.

The application also uses a common library called `Alexandria`. This library is located in the `${server.config.dir}/lib/Alexandria` directory.

Procedure

1. Configure class loading for the application, so that the application can access the third-party APIs.

In the `server.xml` file, or an included file, configure the API type visibility by adding the following code:

```
<application id="scholar" name="Scholar" type="ear" location="scholar.ear">
  <classloader apiTypeVisibility="spec, ibm-api, third-party" commonLibraryRef="Alexandria" />
</application>
```

2. Optional: If the application uses any common libraries, set those libraries to use the same API type visibility setting.

In the `server.xml` file, or an included file, add the following code:

```
<library id="Alexandria" apiTypeVisibility="spec, ibm-api, third-party">
  <fileset dir="${server.config.dir}/lib/Alexandria" includes="*.jar" scanInterval="5s" />
</library>
```

Removing access to third-party APIs for a Java EE application

By default, Java EE applications do not have access to the third-party APIs available in the Liberty profile. You can also configure this state explicitly in the `server.xml` file, or an included file.

About this task

In the following example, an application called `Scholar` had previously been configured to access third-party APIs, as described in “Accessing third-party APIs from a Java EE application” on page 108. You want to remove this access, and to make it explicit in the configuration that the application now uses the default access setting.

The application also uses a common library called `Alexandria`. This library is located in the `${server.config.dir}/lib/Alexandria` directory.

Procedure

1. Configure class loading for the application, to show that the application can no longer access the third-party APIs.

In the `server.xml` file, or an included file, remove `third-party` from the set of values included for the `apiTypeVisibility` attribute:

```
<application id="scholar" name="Scholar" type="ear" location="scholar.ear">
  <classloader apiTypeVisibility="spec, ibm-api" commonLibraryRef="Alexandria" />
</application>
```

2. Optional: If the application uses any common libraries, set those libraries to use the same API type visibility setting.

In the `server.xml` file, or an included file, add the following code:

```
<library id="Alexandria" apiTypeVisibility="spec, ibm-api">
  <fileset dir="${server.config.dir}/lib/Alexandria" includes="*.jar" scanInterval="5s" />
</library>
```

Overriding a provided API with an alternative version

If an application provides (or uses a library that provides) classes that are also available in the Liberty profile, by default the classes from the Liberty profile are used. To change this so that the application uses the alternative versions of these classes, the application must be configured in the `server.xml` file, or an included file.

About this task

If a web application includes classes that are also present in the server runtime environment, you might want to control which copy of each of those classes is used by the application. For example, if different versions of the classes are present in both the application and the server runtime environment, you have to ensure that the version packaged in the application is used.

By default, classes from the Liberty profile runtime environment are used by all Java EE applications. You can override this behavior by using the class loader configuration **delegation** attribute. This configuration is specific to a particular application, or to a shared library that can be selected for use by an application.

Example

In the following example, an application called `Scholar` needs to use classes that it provides (or that are provided in a library that it uses), rather than using the copies of the classes that are available in the Liberty profile.

- When the classes are packaged within the application, override the default `parentFirst` delegation behavior with a `classloader` element in the `server.xml` configuration file or a file that it includes:


```
<application id="" name="Scholar" type="ear" location="scholar.ear">
  <classloader delegation="parentLast" />
</application>
```

This tells the application class loader to look at the Liberty profile classes only after looking in the application and its associated libraries for a class.

- When the classes are packaged in a shared library, add the **delegation** attribute to the `classloader` element that configures the use of the shared library as follows:

```
<application id="" name="Scholar" type="ear" location="scholar.ear">
  <classloader delegation="parentLast" commonLibraryRef="mySharedLib"/>
</application>
```

```
<library id="mySharedLib">
  <fileset dir="${server.config.dir}/myLib" includes="*.jar" />
</library>
```

You can also use the **privateLibraryRef** attribute for private libraries in an application. See “Sharing a library across multiple Java EE applications” on page 107.

Providing global libraries for all Java EE applications

You can provide global libraries that can be used by any application. You do this by putting the JAR files for those libraries in a global library directory, then specifying use of global libraries in the class loader configuration for each application.

About this task

Under the `${WLP_USER_DIR}` directory, there are two locations in which you can place global libraries:

- `shared/lib/global`
- `servers/server_name/lib/global`

If there are files present in these locations at the time an application is started, and that application does not have a `<classloader>` element configured, the application uses these libraries. If a class loader configuration is present, these libraries are not used unless the global library is explicitly referenced.

CAUTION:

If you use global libraries, you are advised also to configure a `<classloader>` element for every application. The servlet specification requires applications to share the global library class loader in their class loader parent chain. This breaks the separation of class loaders for each application that is otherwise possible. Consequently, applications are more likely to have long-lasting effects on classes loaded in Liberty and on each other, and class space consistency issues are more likely to arise between applications, especially over time as features are added and removed in a running server. None of these considerations apply for applications that specify a `<classloader>` element in their configuration, because they maintain this separation.

Example

In the following example, an application called `Scholar` is configured to use a common library called `Alexandria`, and also to use the global library.

In the `server.xml` file, or an included file, enable the global library for an application by adding the following code:

```
<application id="" name="Scholar" type="ear" location="scholar.ear">
  <classloader apiTypeVisibility="spec" commonLibraryRef="Alexandria, global" />
</application>
```


The settings for the global library can also be configured explicitly, as a library element with the special ID `global`. For example:

```
<library id="global">
  <fileset dir="/path/to/folder" includes="*.jar" />
</library>
```

Deploying data access applications to the Liberty profile

Deploying a data access application includes more than installing your web application archive (WAR) or enterprise archive (EAR) file onto a Liberty profile. Deployment can include tasks for configuring the data access resources of the server and overall runtime environment.

About this task

The following topics are covered in this section:

Procedure

- Configure a data source and JDBC driver for database connectivity in a Liberty profile
- Deploy an JDBC application to the Liberty profile
- Optional: Configure connection pooling in the Liberty profile
- Optional: Develop an application-defined data source on the Liberty profile
- Optional: Configure transaction recovery for data sources on the Liberty profile
- Migrating data access applications to the Liberty profile

Deploying an existing JDBC application to the Liberty profile

You can take an existing application that uses Java Database Connectivity (JDBC) and a data source, and deploy the application to a server.

About this task

You can take an existing JDBC application and deploy it to the Liberty profile. To do this, you add the `jdbc-4.0` server feature to the `server.xml` file, along with code that tells the server the JDBC driver location and specifies properties that the JDBC driver uses to connect to the database.

This example uses the `ImpactWeb` sample application. This application includes a servlet called `WorkingServlet`. In this example, you extend the servlet with code that tests that the JDBC application is working as expected.

Procedure

1. Create a server.
2. Add the `jdbc-4.0` and the `servlet-3.0` server features to the `server.xml` file.
3. Add code to the `server.xml` file to specify the database type and the data source location.

For example:

```
<jdbcDriver id="DerbyEmbedded" libraryRef="DerbyLib"/>
<library id="DerbyLib">
  <fileset dir="C:/myDerbyLocation/lib" includes="derby.jar"/>
</library>
<dataSource id="ds1" jndiName="jdbc/exampleDS" jdbcDriverRef="DerbyEmbedded">
  <properties.derby.embedded
    databaseName="C:/myDerbyLocation/data/exampleDB"
    createDatabase="create"
  />
</dataSource>
```

For information about other options for coding data source definitions, see Liberty profile: Using Ref tags in configuration files.

4. Optional: Enable JDBC tracing.
5. Modify the `WorkingServlet.java` servlet.

For example, add the following code:

```
@Resource(name = "jdbc/exampleDS")
DataSource dsl;
Connection con = dsl.getConnection();
Statement stmt = null;
try {
    stmt = con.createStatement();
    // create a table
    stmt.executeUpdate("create table cities
        (name varchar(50) not null primary key, population int, county varchar(30))");
    // insert a test record
    stmt.executeUpdate("insert into cities values ('myHomeCity', 106769, 'myHomeCounty')");
    // select a record
    ResultSet result = stmt.executeQuery("select county from cities where name='myHomeCity'");
    result.next();
    // display the county information for the city.
    System.out.println("The county for myHomeCity is " + result.getString(1));
}
catch (SQLException e) {
    e.printStackTrace();
}
finally {
    try {
        // drop the table to clean up and to be able to rerun the test.
        stmt.executeUpdate("drop table cities");
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    con.close();
}
```

6. Add the application to the server.
7. If it is not already running, start the server.
8. Optional: Test that the JDBC application is working as expected.

For example, run the modified `WorkingServlet.java` servlet. You should see the following console output:

```
[AUDIT ] CWWKZ0001I: The application ImpactWeb has started successfully.
[AUDIT ] CWWKD0000I: The dataSource dsl is available as jdbc/exampleDB.
[AUDIT ] CWWKD0000I: The jdbcDriver DerbyEmbedded is available.
The county for myHomeCity is myHomeCounty
```

Enabling JDBC Tracing for the Liberty profile

JDBC tracing for the Liberty profile is enabled either through a driver-specific custom trace setting, or using the application server supplemental JDBC tracing option.

About this task

There are two ways of using driver-specific custom trace facilities:

- Using the Java built-in logging mechanism, `java.util.logging`, if the driver supports it.
- Configuring a custom trace setting as a vendor property.

If your JDBC driver does not provide its own custom tracing or logging facilities, or the facilities it provides are minimal, you can use supplemental JDBC tracing from the application server.

If you enable tracing by using either a custom vendor property or supplemental JDBC tracing, you must add the logwriter name to the trace specification in the `bootstrap.properties` file. You can use any of the following logwriters:

DB2 `com.ibm.ws.db2.logwriter`

Derby `com.ibm.ws.derby.logwriter`

Informix® JCC (uses the same driver as DB2)

`com.ibm.ws.db2.logwriter`

Informix JDBC

`com.ibm.ws.informix.logwriter`

Microsoft SQL Server JDBC Driver

`com.ibm.ws.sqlserver.logwriter`

DataDirect Connect for JDBC for Microsoft SQL Server

`com.ibm.ws.sqlserver.logwriter`

Sybase

`com.ibm.ws.sybase.logwriter`

Other databases (for example solidDB® and MySQL)

`com.ibm.ws.database.logwriter`

Because changes to trace enablement involve altering the `bootstrap.properties` file, you must restart the server for the changes to take effect.

The following examples illustrate the use of the various JDBC trace methods.

Procedure

- Use `java.util.logging`.

If the driver you are using supports `java.util.logging`, you can enable it by appending the driver's trace level to `com.ibm.ws.logging.trace.specification` in the `bootstrap.properties` file. See [Using Java logging in an application](#), and the JDBC vendor documentation for levels and other trace information specific to your driver.

Here is an example for Microsoft SQL Server JDBC Driver:

- Example code for the `bootstrap.properties` file:

```
com.ibm.ws.logging.trace.specification=*audit=enabled:com.microsoft.sqlserver.jdbc=FINE
```

Here is an example for Oracle JDBC:

- Example code for the `bootstrap.properties` file:

```
com.ibm.ws.logging.trace.specification=*audit=enabled:oracle=FINE
```

- For Oracle, you must also enable the tracing using the system property `oracle.jdbc.Trace`, using one of the following two options:

- In the `bootstrap.properties` file, add the setting `oracle.jdbc.Trace=true`
- In a Java program, add the setting `System.setProperty("oracle.jdbc.Trace","true");`

- Use custom trace settings.

If the driver you are using has custom trace settings, you set them as JDBC driver vendor properties in the `server.xml` file. You also add the logwriter name to the trace specification in the `bootstrap.properties` file.

Here is an example for DB2 JCC, using the custom property `traceLevel`:

- Example code for the `server.xml` file:

```
<dataSource id="db2" jndiName="jdbc/db2" jdbcDriverRef="DB2Driver" >  
  <properties.db2.jcc databaseName="myDB" traceLevel="-1"/>  
</dataSource>
```

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.db2.logwriter=all=enabled

Here is an example for Derby Network Client:

- Example code for the server.xml file:
<dataSource id="derbyNC" jndiName="jdbc/derbyNC" jdbcDriverRef="DerbyNC" >
 <properties.derby.client databaseName="myDB" createDatabase="create" traceLevel="1"/>
</dataSource>

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.derby.logwriter=all=enabled

Here is an example for Informix JCC. This database uses the DB2 drivers for JCC connectivity.

- Example code for the server.xml file:
<dataSource id="informixJCC" jndiName="jdbc/informixJCC" jdbcDriverRef="InformixDriverJCC" >
 <properties.informix.jcc databaseName="myDB" traceLevel="-1"/>
</dataSource>

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.db2.logwriter=all=enabled

- Use supplemental JDBC tracing.

If your JDBC driver does not provide suitable tracing or logging facilities, you can use supplemental JDBC tracing from the application server. The application server automatically determines whether to enable supplemental JDBC tracing, based on the JDBC driver being used. To override this, set the data source property supplementalJDBCTrace to true or false.

1. Enable supplemental tracing.

Here is an example for enabling supplemental tracing with the embedded Derby database. Supplemental JDBC tracing is enabled by default for this database, so you only need to set the logwriter in the bootstrap.properties file:

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.derby.logwriter=all=enabled

Here is an example for enabling supplemental tracing with Informix JDBC. Supplemental JDBC tracing is enabled by default for this database.

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.informix.logwriter=all=enabled

Here is an example for enabling supplemental tracing, and java.util.logging, with Microsoft SQL Server JDBC Driver:

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.sqlserver.logwriter=all=enabled:
com.microsoft.sqlserver.jdbc=all

Here is an example for enabling supplemental tracing with DataDirect Connect for JDBC for Microsoft SQL Server:

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.microsoft.sqlserver.jdbc=all

Here is an example for enabling supplemental tracing with solidDB. Supplemental JDBC tracing is enabled by default for this database.

- Example code for the server.xml file:
<dataSource id="soliddb" jndiName="jdbc/soliddb" jdbcDriverRef="solidDBDriver">
 <properties databaseName="dba" URL="jdbc:solid://localhost:2315/dba/dba" />
</dataSource>

- Example code for the bootstrap.properties file:
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.database.logwriter=all=enabled

Here is an example for enabling supplemental tracing with Sybase. Supplemental JDBC tracing is enabled by default for this database.

- Example code for the bootstrap.properties file:

```
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.sybase.logwriter=all=enabled
```

Here is an example for enabling supplemental tracing with other databases:

- Example code for the bootstrap.properties file:

```
com.ibm.ws.logging.trace.specification==audit=enabled:com.ibm.ws.database.logwriter=all=enabled
```

2. Disable supplemental tracing

To disable supplemental JDBC tracing, either set the supplementalJDBCTrace data source property to false in the server.xml file, or remove the logwriter name from the com.ibm.ws.logging.trace.specification property in the bootstrap.properties file:

- Example code for the server.xml file for solidDB:

```
<dataSource id="soliddb" jndiName="jdbc/soliddb" jdbcDriverRef="solidDBDriver" supplementalJDBCTrace="false">
  <properties databaseName="dba" URL="jdbc:solid://localhost:2315/dba/dba" />
</dataSource>
```

- Example code for the bootstrap.properties file for solidDB:

```
com.ibm.ws.logging.trace.specification==audit=enabled
```

Deploying a web application to the Liberty profile

By deploying a helloworld.war application, you can learn how server configurations change in the Liberty profile.

Before you begin

The helloworld.war application uses a simple servlet to display a message on your browser. You can create any other messages to be displayed. The coding of the application is not described within the Liberty profile documents.

About this task

When you deploy a web application to the Liberty profile when the server is up and running, all configurations related to the application are automatically enabled in the server.xml file. However, you can also configure the server.xml file manually by completing the following steps.

This example uses the helloworld.war application and can be accessed via `http://localhost:9090/helloworld`. In this example, we create a new Liberty profile server instance and change its default HTTP port to 9090, then deploy the application on it.

Procedure

1. Create a server named hserver using the command `server create hserver`.
2. Create a directory apps for application deployment under the newly created server directory. The directory should be like `/usr/servers/hserver/apps`.
3. Copy the helloworld.war application into the apps directory created.
4. Change the default HTTP port of the server hserver to 9090 by adding the following line into the server.xml file.

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9090" />
```

5. Configure the application by updating the server.xml as follows:

```
<server description="Hello World Server">
```

```
  <featureManager>
    <feature>servlet-3.0</feature>
  </featureManager>
```

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9090" />
```

```

<application context-root="helloworld" type="war" id="helloworld"
  location="helloworld.war" name="helloworld"/>

</server>

```

Where **context-root** specifies the entry point of the deployed application. The entry point of an deployed application is determined in the following precedence:

- **context-root** in the server.xml file
 - application.xml , if an EAR application
 - ibm-web-ext.xml, if a web application
 - **name** of the application in the server.xml file, if a web application
 - Manifest.MF, if a WAB application
 - Directory name or the file name relative to the "dropins" directory of the Liberty profile
6. Start the server in foreground using the command `server run hwserver`.
 7. Test the application at `http://localhost:9090/helloworld`.
 8. Optional: Stop the server if you don't need it.

Deploying a JPA application to the Liberty profile

To enable the Liberty profile to support an application that use the Java Persistence API (JPA), you add the `jpa-2.0` feature to the `server.xml` file. You also need to define persistence contexts and persistence units, and configure access to the entity manager and entity manager factory.

Before you begin

This task assumes that you have created a Liberty profile server, on which you want to deploy an application that uses JPA. See [Creating a new Liberty profile server from the command prompt](#).

About this task

The `jpa-2.0` feature provides support for applications that use application-managed and container-managed JPA written to the JPA 2.0 specification. Support is built on top of Apache OpenJPA with extensions to support the container-managed programming model.

Procedure

- Add the `jpa-2.0` feature to the `server.xml` file.
- Add persistence context and persistence unit definitions to the `web.xml` file.

For example:

```

<persistence-context-ref>
  <persistence-context-ref-name>example/em</persistence-context-ref-name>
  <persistence-unit-name>ExamplePersistenceUnit</persistence-unit-name>
</persistence-context-ref>

```

```

<persistence-unit-ref>
  <persistence-unit-ref-name>example/emf</persistence-unit-ref-name>
  <persistence-unit-name>ExamplePersistenceUnit</persistence-unit-name>
</persistence-unit-ref>

```

- Configure access to the entity manager.

For example:

```

Context ctx = new InitialContext();
UserTransaction tran = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
tran.begin();

```

```
EntityManager em = (EntityManager) ctx.lookup(java:comp/env/example/em");
Thing thing = new Thing();
em.persist(thing);
tran.commit();
```








- Configure access to the entity manager factory.

For example:

```
Context ctx = new InitialContext();
EntityManagerFactory emf = (EntityManagerFactory) ctx.lookup(java:comp/env/example/emf");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();
Thing thing = new Thing();
em.persist(thing);
tx.commit();
int id = thing.getId();
em.close();
```

Chapter 11. How do I deploy applications?

Follow these shortcuts to get started quickly with popular tasks.

-  Deploy enterprise application files.
-  Change the configuration of deployed enterprise application files.
-  Configure class loaders.
-  Start or stop deployed enterprise application files.
-  Update deployed enterprise application files.
-  Deploy and administer business-level applications.
-  Troubleshoot deployment problems

Chapter 12. Deploying enterprise applications

Deploying Java Platform, Enterprise Edition (Java EE) application files consists of placing assembled enterprise application, web, enterprise bean (EJB), or other installable modules on a server or cluster configured to hold the files. Installed files that start and run properly are considered *deployed*.

Installing enterprise application files

As part of deploying an application, you install application files on a server configured to hold installable modules.

Before you begin

Before you can install your Java Platform, Enterprise Edition (Java EE) application files on an application server, you must assemble modules as needed.

Also, before you install the files, configure the target application server. As part of configuring the server, determine whether your application files can be installed to your deployment targets.

Note: Installation of new Java EE specification levels are not allowed on deployment targets that do not support the specified Java EE level. If an application contains modules with an EJB deployment descriptor level of 3.1 (part of Java EE 6), the application cannot be installed on an application server that does not support Java EE 6; for example, application server versions prior to Version 8. Further, if an application contains Java EE 6 annotations, the application cannot be installed on an application server that does not support the Java EE 6 specification level, even if the application contains an EJB deployment descriptor for a previous version of the supported Java EE specification level; for example, EJB 3.0 which is part of Java EE 5.

About this task

You can install the following enterprise modules on a server:

- Enterprise archive (EAR)
- Enterprise bean (EJB)
- Web archive (WAR)
- Session Initiation Protocol (SIP) module (SAR)
- Resource adapter (connector or RAR)
- Application client modules

Application client files can be installed in a WebSphere Application Server configuration but cannot be run on a server.

Complete the following steps to install your files.

Procedure

1. Determine which method to use to install your application files. The product provides several ways to install modules.
2. Install the application files using
 - Administrative console
 - wsadmin scripts
 - Java administrative programs that use Java Management Extensions (JMX) application programming interfaces (APIs)
 - Java programs that define a Java EE DeploymentManager object in accordance with Java EE Application Deployment specification (JSR-88)

3. Start the deployed application files using
 - Administrative console
 - `wsadmin startApplication`
 - Java programs that use `ApplicationManager` or `AppManagement MBeans`
 - Java programs that define a Java EE `DeploymentManager` object in accordance with Java EE Application Deployment specification (JSR-88)

What to do next

Save the changes to your administrative configuration.

When saving the configuration in multiple-server environments, synchronize the configuration with the nodes where the application is expected to run.

Note:

- You must use either the administrative console or `wsadmin` scripting to synchronize a node. Of these two options, using the administrative console is the best way to perform this operation. The Nodes panel in the administrative console includes the **Synchronize** operation. If you need to use `wsadmin` scripting to synchronize a node, use the `NodeSync` mbean's `sync()` command.
- Do not restart the node agent as part of the synchronize node process. Administration operations, such as node synchronization for application deployment, or updates that take place while the node agent is starting, that are initiated through the node agent, and affect the application servers, fail until the node agent has a chance to discover the application servers.

Next, test the application. For example, point a web browser at the URL for a deployed application. Typically, the URL is `http://hostname:9060/web_module_name`, where `hostname` is your valid web server and 9060 is the default port number. Examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

If your application contains many classes with annotations and takes a long time to deploy, you can reduce annotation searches to speed up deployment. See the topic on reducing annotation searches during application deployment.

Installable enterprise module versions

The contents of a Java Platform, Enterprise Edition (Java EE) module affect whether you can install the module on a deployment target. A *deployment target* is a server or a cluster with at least one member on a WebSphere Application Server product.

Installable application modules

Select only appropriate deployment targets for a module. You must install an application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) archive (SAR), web module, or client module on a Version 8.x target under any of the following conditions:

- The module supports Java Platform, Enterprise Edition (Java EE) 6.
- The module calls an 8.x runtime application programming interface (API).
- The module uses an 8.x product feature.

For example, because support for deployment of application client modules using the administrative console or `wsadmin AdminApp` commands was added in Version 8.0, you must install a client module using the console or an `AdminApp` command only to a Version 8.x target.

If a module supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you can install the module on a Version 6.x, 7.x or 8.x deployment target. Modules that call a 6.1.x API or use a 6.1.x feature can be installed on a 6.1.x, 7.x or 8.x deployment target. Modules that call a 6.0.x API or use a 6.0.x feature can

be installed on a 6.0.x, 6.1.x, 7.x or 8.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 7.x or 8.x deployment target or on a 6.1.x deployment target that has been enabled with that feature pack. Modules that require 7.x feature pack functionality can be installed on a 8.x deployment target or on a 7.x deployment target that has been enabled with that feature pack.

Selecting options such as **Precompile JavaServer Pages files**, **Deploy web services** or **Deploy enterprise beans** during application installation indicates that the application uses features of the current product version. Do not select these options if the targets of the applications are on older version nodes. Use the tools available in the older version, such as JspBatchCompiler, wsgen, or ejbdeploy, to update your application with generated code before deploying your application.

Note: You must package container-managed persistence (CMP) or bean-managed persistence (BMP) entity beans in an EJB 2.1 or earlier module. You cannot install an EJB 3.0 or EJB 3.1 module that contains CMP or BMP entity beans. Installation fails when a CMP or BMP entity bean is packaged in an EJB 3.0 or EJB 3.1 module. You can install EJB 2.1 or earlier modules on a 6.x, 7.x or 8.x deployment target.

Installable RAR files

You can install a stand-alone resource adapter (connector) module, or RAR file, developed for a Version 6.0.x product to a 6.x, 7.x or 8.x deployment target. If the module calls a 6.1.x API, then you must install the module on a 6.1.x, 7.x or 8.x deployment target. You must install a module that calls a 7.x API on a 7.x or 8.x deployment target. You must install a module that calls a 8.x API on a 8.x deployment target.

Deployment targets

The following table lists the compatible deployment target versions for various modules. “6.x, 7.x or 8.x” for Deployment target versions indicates that you can deploy the module to a WebSphere Application Server Version 6, 7, or 8 server or cluster.

Table 16. Compatible deployment target versions for 6.x, 7.x and 8.x modules. Deploy modules to compatible deployment target versions.

Module type	Module Java support	Module calls 6.x, 7.x or 8.x runtime APIs or uses 6.x, 7.x or 8.x features?	Client versions that can install module	Deployment target versions
Application, EJB, or web	J2EE 1.3	No	6.x, 7.x or 8.x	6.x, 7.x or 8.x
Application, EJB, or web	J2EE 1.3	Yes	6.x, 7.x or 8.x for 6.x APIs or features 7.x or 8.x for 7.x APIs or features 8.x for 8.x APIs or features	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs or use 6.1.x features on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs or use 6.0.x features on any 6.x, 7.x or 8.x deployment target.
Application, EJB, SAR, or web	J2EE 1.4	Yes or No	6.x, 7.x or 8.x	6.x, 7.x or 8.x
Application, EJB, SAR, or web	Java EE 5	Yes or No	7.x or 8.x	7.x or 8.x

Table 16. Compatible deployment target versions for 6.x, 7.x and 8.x modules (continued). Deploy modules to compatible deployment target versions.

Module type	Module Java support	Module calls 6.x, 7.x or 8.x runtime APIs or uses 6.x, 7.x or 8.x features?	Client versions that can install module	Deployment target versions
Application, EJB, SAR, or web	Java EE 6	Yes or No	8.x	8.x
Client	Any Java EE version	Yes or No	8.x	8.x
Resource adapter	JCA 1.0	No	6.x, 7.x or 8.x	6.x, 7.x or 8.x
Resource adapter	JCA 1.0	Yes	6.x, 7.x or 8.x	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x, 7.x or 8.x deployment target.
Resource adapter	JCA 1.5	Yes or No	6.x, 7.x or 8.x	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x, 7.x or 8.x deployment target.
Resource adapter	JCA 1.6	Yes or No	JCA 1.6 resource adapters can only be installed on 8.x. Resource adapter archive annotations are not supported on previous WebSphere Application Server releases.	JCA 1.6 resource adapters can only be installed on 8.x. Resource adapter archive annotations are not supported on previous WebSphere Application Server releases.

Ways to install enterprise applications or modules

The product provides several ways to install Java Platform, Enterprise Edition (Java EE) application files.

Installable files include enterprise archive (EAR), enterprise bean (EJB), web application archive (WAR), Session Initiation Protocol (SIP) archive (SAR), resource adapter (connector or RAR), and application client modules. They can be installed on a server or cluster. Application client files can be installed in a WebSphere Application Server configuration but cannot be run on a server.

Table 17. Ways to install application files. Deploy an application or module using the administrative console, wsadmin, programming, or deployment tools.

Option	Method	Modules	Comments	Starting after install
Administrative console install wizard See topics on installing enterprise application files with the console.	Click Applications > New application > New Enterprise Application in the console navigation tree and follow instructions in the wizard.	Files for all of the following modules: • EAR • EJB • WAR • SAR • RAR • Application client	Provides one of the easier ways to install application files. For applications that do not require changes to the default bindings, after you specify the application file, expand Choose to generate default bindings and mappings , select Generate default bindings , click the Summary step, and then click Finish .	Click Start on the Enterprise applications page accessed by clicking Applications > Application Types > WebSphere enterprise applications in the console navigation tree.
wsadmin scripts	Invoke AdminApp object install commands in a script or at a command prompt.	Files for all of the following modules: • EAR • EJB • WAR • SAR • RAR • Application client	"Getting started with scripting" in the Using the administrative clients PDF provides an overview of wsadmin.	<ul style="list-style-type: none"> Invoke the AdminApp startApplication command. Invoke the startApplication method on an ApplicationManager MBean using AdminControl.
Job manager runs wsadmin scripts	Invoke AdminTask.submitJob -jobType installApplication command in a script or at a command prompt.	Files for all of the following modules: • EAR • EJB • WAR • SAR • RAR • Application client	Use the job manager to submit a job that installs the application. You can also submit jobs that start and stop the application at designated times. "Administering jobs in a flexible management environment using scripting" in the Using the administrative clients PDF provides an overview of the job manager.	<ul style="list-style-type: none"> Invoke the AdminTask.submitJob -jobType startApplication command. Invoke the AdminApp startApplication command. Invoke the startApplication method on an ApplicationManager MBean using AdminControl.
Java application programming interfaces	Install programs by completing the steps in Installing an application through programming.	All EAR files	Use Java Management Extensions (JMX) MBeans to install the application. For an overview of Java MBean programming, see Managing applications through programming.	Start the application by calling the startApplication method on a proxy.
Rapid deployment tools Refer to topics under Rapid deployment of J2EE applications .	Briefly, do the following: 1. Update your J2EE application files. 2. Set up the rapid deployment environment. 3. Create a free-form project. 4. Launch a rapid deployment session. 5. Drop your updated application files into the free-form project.	J2EE modules at the J2EE 1.3 or 1.4 specification levels, including EAR files and the following stand-alone modules: • EJB • WAR • SAR • RAR • Application client The rapid deployment tools do not support the J2EE 1.2 or Java EE 5.0 and later specification levels. Use this option for drag and drop deployment of J2EE 1.3 or 1.4 modules. Unlike the monitored directory option, the rapid deployment tools do not support drag and drop deployment of Java EE 5.0 and later modules.	Rapid deployment tools offer the following advantages: <ul style="list-style-type: none"> You do not need to assemble your J2EE application files prior to deployment. You do not need to use other installation tools mentioned in this table to deploy the files. For a list of ways in which the rapid deployment tools differ from monitored directory deployment, see the monitored directory description in this table.	Use any of the options in this table to start the application. Clicking Start on the Enterprise applications page is the easiest option.

Table 17. Ways to install application files (continued). Deploy an application or module using the administrative console, wsadmin, programming, or deployment tools.

Option	Method	Modules	Comments	Starting after install
Java programs	Code programs that use Java EE DeploymentManager (JSR-88) methods. Note: Application installation using JSR-88 was deprecated in WebSphere Application Server Version 8.0. Use another way listed in this table to deploy applications or modules.	All Java EE modules, including EAR files and the following stand-alone modules: <ul style="list-style-type: none"> • EJB • WAR • SAR • RAR • Application client 	<ul style="list-style-type: none"> • Uses Java EE Application Deployment Specification (JSR-88). • Can customize modules using DConfigBeans. 	Call the Java EE DeploymentManager (JSR-88) start method in a program to start the deployed modules when the module's running environment initializes.

Installing enterprise application files with the console

Installing Java Platform, Enterprise Edition (Java EE) application files consists of placing assembled enterprise application, Web, enterprise bean (EJB), or other installable modules on a server or cluster configured to hold the files. Installed files that start and run properly are considered *deployed*.

Before you begin

Before installing enterprise application files, ensure that you are installing your application files onto a compatible deployment target. If the deployment target is not compatible, select a different target.

Optionally, determine whether the application that you are installing uses library files that other deployed applications also use. You can define a shared library for each of these shared files. Using shared libraries reduces the number of library file copies on your workstation or server.

About this task

To install new enterprise application files to a WebSphere Application Server configuration, you can use the following options:

- Administrative console
- wsadmin scripts
- Monitored directory deployment
- Application properties files
- Java MBean programs
- Java programs that call Java EE DeploymentManager (JSR-88) methods

This topic describes how to use the administrative console to install an application, EJB component, Session Initiation Protocol (SIP) archive (SAR), or web module.

Note: After you start completing steps in the application installation wizard, click **Cancel** to exit if you decide not to install the application. Do not simply move to another administrative console page without first clicking **Cancel** on an application installation page.

Procedure

1. Click **Applications > New application > New Enterprise Application** in the console navigation tree.
2. On the first Preparing for application installation page:
 - a. Specify the full path name of the source enterprise application file (.ear file otherwise known as an *EAR file*).

The EAR file that you are installing can be either on the client machine (the machine that runs the Web browser) or on the server machine (the machine to which the client is connected). If you specify an EAR file on the client machine, then the administrative console uploads the EAR file to the machine on which the console is running and proceeds with application installation.

You can also specify a stand-alone web archive (WAR), SAR, or Java archive (JAR) file for installation.

IBM i If the EAR file resides on the server machine, and the server is an IBM i server, ensure that user profile QEJBSVR has *R authority to the EAR file and at least *X authority to all the directories in the path containing the EAR file.

- b. Click **Next**.
3. On the second Preparing for application installation page:
 - a. Select whether to view all installation options.

Fast Path - Prompt only when additional information is required
Displays the module mapping step as well as any steps that require you to specify needed information to install the application successfully.

Detailed - Show all installation options and parameters
Displays all installation options.
 - b. Select whether to generate default bindings.

Select **Generate default bindings** to have the product supply default values for incomplete Java Naming and Directory (JNDI) and other application bindings. The product does not change existing bindings.

You do not need to specify JNDI values for EJB bean, local home, remote home, or business interfaces of EJB 3.x modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.x or a Web 2.4 or later module, you do not need to specify JNDI values because the product resolves the targets automatically during run time. Even when you select **Generate default bindings**, the product does not generate default values for those JNDI values but it does generate default values for other bindings such as virtual host.

You can customize default values used in generating default bindings. “Preparing for application installation binding settings” on page 136 describes available customization and provides sample bindings.
 - c. Click **Next**. If security warnings are displayed, click **Continue**. The Install New Application pages are displayed. If you chose to generate default bindings, you can proceed to the Summary step. “Example: Installing an EAR file using the default bindings” on page 133 provides sample steps.
4. Specify values for installation options as needed.

You can click on a step number to move directly to that page instead of clicking **Next**. The contents of the application or module that you are installing determines which pages are available.

Table 18. Wizard page descriptions. The table describes each wizard page.

Page	Description
Select installation options	On the Select installation options page, provide values for the settings specific to the product. Default values are used if you do not specify a value.
Map modules to servers	<p>On the Map modules to servers page, specify deployment targets where you want to install the modules contained in your application. Modules can be installed on the same deployment target or dispersed among several deployment targets. Each module must be mapped to a target server.</p> <p>On single-server products, a deployment target can be an application server or web server.</p> <p>On multiple-server products, a deployment target can be an application server, cluster of application servers or web server.</p>
Provide options to compile JSPs	If the Precompile JavaServer Pages files setting is enabled on the Select installation options page and your application uses JavaServer Pages (JSP) files, then you can specify JSP compiler options on the Provide options to compile JSPs page.

Table 18. Wizard page descriptions (continued). The table describes each wizard page.

Page	Description
Provide JNDI names for beans	<p>On the Provide JNDI names for beans page, specify a JNDI name for each enterprise bean in every EJB 2.1 and earlier module. You must specify a JNDI name for every enterprise bean defined in the application. For example, for the EJB module <code>MyBean.jar</code>, specify <code>MyBean</code>.</p> <p>As to EJB 3.x modules, you can specify JNDI names, local home JNDI names, remote home JNDI names, or no JNDI names. If you do not specify a value, the product provides a default value.</p>
Bind EJB business	<p>On the Bind EJB business page, you can specify business interface JNDI names for EJB 3.x modules. If you specified a JNDI name for a bean on the Provide JNDI names for beans page, do not specify a business interface JNDI name on this page for the same bean. If you do not specify the JNDI name for a bean, you can optionally specify a business interface JNDI name. When you do not specify a business interface JNDI name, the product provides a container default. For a no-interface view, the business interface value is an empty string ("").</p>
Map default data sources for modules containing 1.x entity beans	<p>If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for Map default data sources for modules containing 1.x entity beans, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.</p>
Map EJB references to beans	<p>On the Map EJB references to beans page, if your application defines EJB references, you can specify JNDI names for enterprise beans that represent the logical names specified in EJB references.</p> <p>If the EJB reference is from an EJB 3.x, or Web 2.4 or later module, the JNDI name is optional. For earlier modules, each EJB reference defined in the application must be bound to an EJB file.</p> <p>If Allow EJB reference targets to resolve automatically is enabled, the JNDI name is optional for all modules. The product provides a container default value or automatically resolves the EJB reference for incomplete bindings.</p>
Map resource references to resources	<p>If your application defines resource references, for Map resource references to resources, specify JNDI names for the resources that represent the logical names defined in resource references. You can optionally specify login configuration name and authentication properties for the resource. After specifying authentication properties, click OK to save the values and return to the mapping step. You can optionally specify extended data source properties to enable a data source that uses heterogeneous pooling to connect to a DB2 database. Each resource reference defined in the application must be bound to a resource defined in your WebSphere Application Server configuration before clicking Finish on the Summary page.</p>
Map virtual hosts for web modules	<p>If your application uses web modules, for Map virtual hosts for web modules, select a virtual host from the list to map to a web module defined in the application. The port number specified in the virtual host definition is used in the URL that is used to access artifacts such as servlets and JSP files in the web module. Each web module must have a virtual host to which it maps. Not specifying all needed virtual hosts will result in a validation error displaying after you click Finish on the Summary page.</p>

Table 18. Wizard page descriptions (continued). The table describes each wizard page.

Page	Description
Map security roles to users or groups	<p>If the application has security roles defined in its deployment descriptor then, for Map security roles to users or groups, specify users and groups that are mapped to each of the security roles. Select Role to select all the roles or select individual roles. For each role, you can specify whether predefined users such as Everyone or All authenticated users are mapped to it. To select specific users or groups from the user registry:</p> <ol style="list-style-type: none"> 1. Select a role and click Lookup users or Lookup groups. 2. On the Lookup users or groups page displayed, enter search criteria to extract a list of users or groups from the user registry. 3. Select individual users or groups from the results displayed. 4. Click OK to map the selected users or groups to the role selected on the Map security roles to users or groups page.
Map RunAs roles to users	<p>If the application has Run As roles defined in its deployment descriptor, for Map RunAs roles to users, specify the Run As user name and password for every Run As role. Run As roles are used by enterprise beans that must run as a particular role while interacting with another enterprise bean. Select Role to select all the roles or select individual roles. After selecting a role, enter values for the user name, password, and verify password and click Apply.</p>
Ensure all unprotected 1.x methods have the correct level of protection	<p>If your application contains EJB 1.x CMP beans that do not have method permissions defined for some of the EJB methods, for Ensure all unprotected 1.x methods have the correct level of protection, specify if you want to leave such methods unprotected or assign protection with deny all access.</p>
Bind listeners for message-driven beans	<p>If your application contains message driven enterprise beans, for Bind listeners for message-driven beans, provide a listener port name or an activation specification JNDI name for every message driven bean.</p>
Map default data sources for modules containing 2.x entity beans	<p>If your application uses EJB modules that contain CMP beans that are based on the EJB 2.x specification, for Map default data sources for modules containing 2.x entity beans, specify a JNDI name for the default data source and the type of resource authorization to be used for the default data source for the EJB modules. You can optionally specify a login configuration name and authentication properties for the data source. When creating authentication properties, you must click OK to save the values and return to the mapping step. You can optionally specify extended data source properties to enable a data source that uses heterogeneous pooling to connect to a DB2 database. The default data source for EJB modules is optional if data sources are specified for individual CMP beans.</p>
Map data sources for all 2.x CMP beans	<p>If your application has CMP beans that are based on the EJB 2.x specification, on the Map data sources for all 2.x CMP beans page, for each of the 2.x CMP beans specify a JNDI name and the type of resource authorization for data sources to be used.</p> <p>You can optionally specify a login configuration name and authentication properties for the data source. When creating authentication properties, you must click OK to save the values and return to the mapping step. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If a default data source for the EJB module and a data source for individual CMP beans are not specified, then a validation error is displayed after you click Finish and installation is canceled.</p>
Ensure all unprotected 2.x methods have the correct level of protection	<p>If your application contains EJB 2.x CMP beans that do not have method permissions defined in the deployment descriptors for some of the EJB methods, on the Ensure all unprotected 2.x methods have the correct level of protection page, specify whether you want to assign a specific role to the unprotected methods, add the methods to the exclude list, or mark them as deselected. Methods added to the exclude list are marked as uncallable. For methods marked deselected no authorization check is performed before their invocation.</p>

Table 18. Wizard page descriptions (continued). The table describes each wizard page.

Page	Description
Provide options to perform the EJB Deploy	<p>If the Deploy enterprise beans setting is enabled on the Select installation options page, then you can specify options for the EJB deployment tool on the Provide options to perform the EJB Deploy page. On this page, you can specify extra class paths, RMIC options, database types, and database schema names to be used while running the EJB deployment tool.</p> <p>You can specify the EJB deployment tool options on this page when installing or updating an application that contains EJB modules. The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.x modules.</p>
Map shared libraries	<p>On the Shared library references and Shared library mapping pages, specify shared library files for your application or web modules to use. A defined shared library must exist to associate your application or module to the library file.</p>
Map shared library relationships	<p>On the Map shared library relationships page, specify relationship identifiers and composition unit names for shared libraries that modules in your enterprise application reference.</p> <p>When installing your enterprise application, the product creates a composition unit for each shared library relationship in the business-level application that you specified for Business-level application name on the Select installation options page.</p>
Provide JSP reloading options for web modules	<p>If your application uses web modules, for Provide JSP reloading options for web modules, configure the class reloading of JavaServer Pages (JSP) files.</p>
Map context roots for web modules	<p>If your application uses web modules that are defined in the application XML deployment descriptor, for Map context roots for web modules, specify a context root for each web module in the application.</p> <p>The product does not include web modules from annotations on this page.</p>
Initialize parameters for servlets	<p>If your application uses web modules that support Servlet 2.5, for Initialize parameters for servlets, specify or override initial parameters that are passed to the init method of web module servlet filters.</p> <p>This page shows servlets from the module XML deployment descriptor. Servlet deployment information from annotations is not available on this page.</p>
Map environment entries for EJB modules	<p>If your application uses EJB modules, for Map environment entries for EJB modules, configure the environment entries of EJB modules such as entity, session, or message driven beans.</p>
Map environment entries for client modules	<p>If you are deploying one or more application client modules, for Map environment entries for client modules, configure the environment entries of client modules that are deployed as JAR files. To view the Map environment entries for client modules page, select the Deploy client modules option on the Select installation options page.</p>
Map environment entries for web modules	<p>If your application uses web modules that support Servlet 2.5, for Map environment entries for web modules, configure the environment entries of web modules such as servlets and JSP files.</p>
Map environment entries for application level	<p>If your application defines one or more environment entries, for Map environment entries for application level, configure the environment entries of applications that are deployed as EAR files.</p>
Map resource environment entry references to resources	<p>If your application contains resource environment references, for Map resource environment entry references to resources, specify JNDI names of resources that map to the logical names defined in resource environment references. If each resource environment reference does not have a resource associated with it, after you click Finish a validation error is displayed.</p>

Table 18. Wizard page descriptions (continued). The table describes each wizard page.

Page	Description
Correct use of system identity	If your application defines Run-As Identity as <i>System Identity</i> , for Correct use of system identity , you can optionally change it to <i>Run-As role</i> and specify a user name and password for the Run As role specified. Selecting <i>System Identity</i> implies that the invocation is done using the WebSphere Application Server security server ID and should be used with caution as this ID has more privileges.
Correct isolation levels for all resource references	If your application has resource references that map to resources that have an Oracle database doing backend processing, for Correct isolation levels for all resource references , specify or correct the isolation level to be used for such resources when used by the application. Oracle databases support ReadCommitted and Serializable isolation levels only.
Map JASPI Provider	On the Map JASPI Provider page, if your application has web modules, you can specify values to override the JASPI settings from the global or domain security configuration. By default, an application inherits the JASPI settings defined in the WebSphere Application Server global or domain security configuration, and web modules inherit the application setting.
Bind message destination references to administered objects	<p>If your application uses message driven beans, for Bind message destination references to administered objects, specify the JNDI name of the J2C administered object to bind the message destination reference to the message driven beans.</p> <p>If the message destination reference is from an EJB 3.0 or later module, then the JNDI name is optional and the run time provides a container default value.</p> <p>Attention: If multiple message destination references link to the same message destination, only one JNDI name is collected. When a message destination reference links to the same message destination as a message driven bean and the destination JNDI name has been collected already, the destination JNDI name for the message destination reference is not collected.</p>
Provide JNDI names for JCA objects	If your application contains an embedded .rar file, for Provide JNDI names for JCA objects , specify the name and JNDI name of each JCA connection factory, administered object and activation specification.
Bind J2C activationspecs to destination JNDI names	If your application contains an embedded .rar file, its activationSpec property has the value <i>Destination</i> , and its introspected type is <i>javax.jms.Destination</i> , for Bind J2C activationspecs to destination JNDI names , specify the <i>jndiName</i> value for each activation bound to it.
Select current backend ID	<p>If your application has EJB modules for which deployment code has been generated for multiple backend databases using an assembly tool, for Select current backend ID, specify the backend ID representing the backend database to be used when the EJB module runs.</p> <p>For information about backend databases, see topics on the EJB deployment tool.</p> <p>This step is not shown if the Deploy enterprise beans setting is enabled on the Select installation options page and if a database type other than None is specified on the Provide options to perform the EJB Deploy page.</p>
Metadata for modules	If your application has EJB 3.x or Web 2.5 modules, you can lock deployment descriptors for one or more of the EJB 3.x or Web 2.5 modules. If you set the <i>metadata-complete</i> attribute to <i>true</i> and lock deployment descriptors, the product writes the complete module deployment descriptor, including deployment information from annotations, to XML format.

Table 18. Wizard page descriptions (continued). The table describes each wizard page.

Page	Description
Provide options to perform the web services deployment	If the Deploy web services setting is enabled on the Select installation options page and your application uses web services, then you can specify wsdeploy command options on the Provide options to perform the web services deployment page. For information about this page, refer to descriptions of the wsdeploy -cp and -jardir options.
Configure remote request dispatch properties	If you are using a remote request dispatcher, you need to first configure it for use. Remote Request Dispatcher (RRD) is a pluggable extension to the web container that allows application frameworks, servlets, and JavaServer Pages to include content from outside of the currently running resource Java virtual machine (JVM) as part of the response sent to the client. Refer to Remote dispatcher property settings for information about where to configure these properties in the administration console.
Display module build ID	If the MANIFEST.MF file of a module in an enterprise application specifies a build identifier, this page shows the build identifier of the module.

5. On the Summary page, verify the cell, node, and server onto which the application modules will install:
 - a. Beside **Cell/Node/Server**, click **Click here**.
 - b. Verify the settings.
 - c. Return to the Summary page.
 - d. Click **Finish**.

Results

Several messages are displayed, indicating whether your application file is installing successfully.

If **Validate input off/warn/fail** on the **Select installation options** page is set to **warn**, the default, several validation warnings might be displayed. If the setting is **fail**, the validation warnings might cause errors.

If you receive an OutOfMemory error and the source application file does not install, your system might not have enough memory or your application might have too many modules in it to install successfully onto the server. If lack of system memory is not the cause of the error, package your application again so the .ear file has fewer modules.

If lack of system memory and the number of modules are not the cause of the error, check the options you specified on the Java virtual machine page of the application server running the administrative console. You might increase the maximum heap size. Then, try installing the application file again.

What to do next

After the application file installs successfully, do the following:

1. Save the changes to your configuration.

For example, click the **Save** link in the application installation messages.

The application is registered with the administrative configuration and application files are copied to the target directory, which is `app_server_root/installedApps/cell_name` by default or the directory that you designate.

For a single-server product, application files are copied to the destination directory when the changes are saved.

For a multiple-server product, files are copied to remote nodes when the configuration on the deployment manager synchronizes with the configuration on individual nodes.

If you clicked the **Save** link in the application installation messages, the Preparing for the application installation page displays again. Click **Applications > Application Types > WebSphere enterprise applications** to exit the page and to see your application in the list of installed applications.

2. Start the application.
3. Test the application. For example, point a web browser at the URL for the deployed application and examine the performance of the application. If necessary, edit the application configuration.

Example: Installing an EAR file using the default bindings

If application bindings were not specified for all enterprise beans or resources in an enterprise application during application development or assembly, you can select to generate default bindings. After application installation, you can modify the bindings as needed using the administrative console.

Before you begin

This topic assumes that the application can run on a web server.

About this task

This topic describes how to install a simple .ear file using the default bindings. You can follow the steps to install any application, including applications provided from the Samples information center.

Procedure

1. Click **Applications > New Application > New Enterprise Application** in the console navigation tree.
2. On the first Preparing for application install page, specify the full path name of the EAR file.

- a. For **Path to the new application**, specify the full path name of the .ear file. For this example, the base file name is my_app1.ear and the file resides on a server in the sample_apps directory.

IBM i For this example, the base file name is my_app1.ear and the file resides on a server at /home/myuserid/myapps. Thus, enter the fully qualified path name for the file, /home/myuserid/myapps/my_app1.ear.

Optionally, select **Remote file system** and click **Browse**. On the Browse Remote Filesystems page, select the node that runs on the server which holds my_app1.ear and the EAR file name.

IBM i Assuming the node is MYISERIES, click **MYISERIES, home, * myuserid, myapps, my_app1.ear ***, and then **OK**.

- b. Click **Next**.
3. On the second Preparing for application install page, choose to generate default bindings.
 - a. Expand **Choose to generate default bindings and mappings**.
 - b. Select **Generate default bindings**.

Using the default bindings causes any incomplete bindings in the application to be filled in with default values. The product does not change existing bindings. By choosing this option, you can skip many of the steps of the application installation wizard and go directly to the **Summary** step.

- c. Click **Next**.
4. If application security warnings are displayed, read the warnings and click **Continue**.
5. On the Install New Application page, click the step number for **Map modules to servers**, and verify the cell, node, and server onto which the application files will install.
 - a. From the **Clusters and servers** list, select the server onto which the application files will install.
 - b. Select all of the application modules.
 - c. Click **Next**.

On the **Map modules to servers** page, you can map modules to other servers such as web servers. If you want a web server to serve the application, use the **Ctrl** key to select an application server or

cluster and the web server together in order to have the plug-in configuration file `plugin-cfg.xml` for that web server generated based on the applications which are routed through it.

6. On the Install New Application page, click the step number beside **Summary**, the last step.
7. On the Summary page, click **Finish**.

What to do next

Examine the application installation progress messages. If the application installs successfully, save your administrative configuration. You can now see the name of your application in the list of deployed applications on the Enterprise applications page accessed by clicking **Applications > Application Types > WebSphere enterprise applications** in the console navigation tree.

If the application does not install successfully, read the messages to identify why the installation failed. Correct problems with the application as needed and try installing the application again.

If the application has a web module, try opening a browser on the application.

1. Point a web browser at the URL for the deployed application.
The URL typically has the format `http://host_name:9060/web_module_name`, where `host_name` is your valid web server and 9060 is the default port number.
2. Examine the performance of the application.

If the application does not perform as desired, edit the application configuration, then save and test it again.

Example: Installing a web services sample with the console

The product provides a web services sample application that you can install on an application server.

Before you begin

Download and extract the `JaxWSServicesSample` sample application. Ensure that your product installation has a Version 7.x or later application server onto which you can install the Web Services Sample.

About this task

The `JaxWSServicesSamples.ear` enterprise application and supporting Java archives (JAR) files are located in the `installableApps` directory within the `JaxWSServicesSamples` sample application.

This topic describes how to install and start the `JaxWSServicesSamples.ear` enterprise application using an administrative console.

Procedure

1. Click **Applications > New Application > New Enterprise Application** in the console navigation tree.
2. On the first Preparing for the application installation page, specify to install `JaxWSServicesSamples.ear`.
 - a. Click **Local file system** or **Remote file system** and specify the full path name of the `JaxWSServicesSamples.ear` file.
`.../installableApps/JaxWSServicesSamples.ear`
 - b. Click **Next**.
3. On the second Preparing for the application installation page, select the fast path option.
 - a. Select **Fast Path - Prompt only when additional information is required**.
 - b. Click **Next**.
4. Click **Next** on each page until you reach the Summary page.

Do not go directly from Step 1 to the Summary page. You must click **Next** on each page that has mandatory settings to enter values for those settings. Simply click **Next** to enter the default values. You optionally can change the values to suit your environment.

5. On the Summary page, verify the cell, node, and server onto which the application modules will install, and then click **Finish**.
6. Examine the application installation progress messages.
If the application installs successfully, the message Application JaxWSServicesSamples installed successfully is displayed. Click **Save**. After the configuration changes are saved, you can see the name of the application in the list of deployed applications on the Enterprise applications page accessed by clicking **Applications > Application Types > WebSphere enterprise applications** in the console navigation tree.

If the application does not install successfully, read the messages to identify why the installation failed. Correct problems with the server or application and try installing the application again.

Results

The **JaxWSServicesSamples** application is in the list of deployed applications on the *Enterprise applications* page.

What to do next

After the application installs successfully, do the following:

1. Start the application.
On the Enterprise applications page, select the check boxes beside **JaxWSServicesSamples**, and then click **Start**.
2. Test the application. Point your web browser at:
`http://localhost:9080/wssamplesei/demo`
If the localhost address does not load, substitute the host name (IP address) of the computer for localhost; for example, `http://9.22.33.44:9080/wssamplesei/demo`.
If you have another WebSphere Application Server installation on your machine, the server port number is likely not 9080. See the Ports table in the administrative console to find the WC_defaulthost server port number. Click **Servers > Server Types > WebSphere application servers > server1 > Ports**. The Port descriptions table lists the important ports.

Table 19. Port descriptions. Use the WC_defaulthost port in the URL to test the sample.

Port name	Description
WC_adminhost	Port used to open an unsecure administrative console in the URL <code>http://host_name:administrative_port/ibm/console</code>
WC_adminhost_secure	Port used to open a secure administrative console in the URL <code>http://host_name:administrative_port/ibm/console</code>
WC_defaulthost	Port used to test running applications in the URL <code>http://host_name:server_port/servlet_name</code>
WC_defaulthost_secure	Port used to securely test running applications in the URL <code>http://host_name:server_port/servlet_name</code>

Preparing for application installation settings

Use this page to specify an application or module to install.

To view this administrative console page, click **Applications > New application > New Enterprise Application**.

This page is the first Preparing for the application installation page. On this page, specify an application or module to install. You can install an enterprise application archive (EAR file), enterprise bean (EJB) module (JAR file), Session Initiation Protocol (SIP) module (SAR file), or web module (WAR file).

The second Preparing for the application installation page has more installation options, such as to generate default bindings for incomplete existing bindings in your application or module.

Path to the new application

Specifies the fully qualified path to the enterprise application file.

The file can be an .ear, .jar, .sar, or .war file.

During application installation, the product typically uploads application files from a client workstation running the browser to the server running the administrative console, and then deploys the application files on the server. In such cases, use the web browser running the administrative console to select EAR, WAR, SAR, or JAR modules to upload to the server.

Use **Local file system** when the browser and application files are on the same computer.

Use **Remote file system** in the following situations:

- The application file resides on any node in the current cell context. Only .ear, .jar, .sar, or .war files are shown during the browsing.
- The application file resides on the file system of any of the nodes in a cell.
- The application file already resides on the computer running the application server. For example, the field value might be `profile_root/installableApps/test.ear`.

After the product transfers the application file, the **Remote file system** value shows the path of the temporary location on the server.

Preparing for application installation binding settings

Use this page to select whether to view all installation options and to change the existing bindings for you application or module during installation. You can choose to generate default bindings for any incomplete bindings in the application or module or to assign specific bindings during installation.

This page is the second Preparing for the application installation page.

To view this administrative console page, click **Applications > New application > New Enterprise Application**, specify the path for the application or module to install, and then click **Next**.

The console page might not display all of the binding options listed in this topic. The contents of the application or module that you are installing determines which options are displayed on the console page. Also, the **Specify bindings to use** option displays only when updating an installed application.

How do you want to install the application?

Specifies whether to show only installation options that require you to supply information or to show all installation options.

Table 20. Installation option descriptions. You can select a Fast Path or select to see all installation options and parameters.

Option	Description
Fast Path - Prompt only when additional information is required	Displays only those options that require your attention, based on the contents of your application or module. Use the fast path to install your application more easily because you do not need to examine all available installation options.

Table 20. Installation option descriptions (continued). You can select a Fast Path or select to see all installation options and parameters.

Option	Description
Detailed - Show all installation options and parameters	Displays all available installation options.

Specify bindings to use

Specifies whether to merge bindings when you update applications or to use new or existing bindings.

This setting is shown only when you update an installed application, and not when you install a new application.

Table 21. Binding option descriptions. You can use merged, new, or existing bindings.

Option	Description
Merge new and existing bindings	The binding information from the updated application or modules is preferred over the corresponding binding information from the installed version. If any element of the binding is missing in the updated version, the corresponding element from the installed version is used. If both the installed and the updated application or module does not have a binding value, the default value is used. The product assigns a default value only if you select the Generate default bindings option.
Use new bindings	The binding information in the updated application or module is used. The binding information from the updated version of the application or module is preferred over the corresponding binding information in the installed version. The binding information from the installed version of the application or module is ignored.
Use existing bindings	The binding information from the installed version of the application or module is preferred over the corresponding binding information from the updated version. If any element of the binding information does not exist in the installed version, the element from the updated version is used. That is, bindings from the updated version of the application or module are ignored if a binding exists in the installed version. Otherwise, the new bindings are honored and not ignored.

Generate default bindings

Specifies whether to generate default bindings and mappings. To view this setting, expand **Choose to generate default bindings and mappings**. If you select **Generate default bindings**, then the product completes any incomplete bindings in the application with default values. The product does not change existing bindings.

After you select **Generate default bindings**, you can advance directly to the Summary step and install the application if none of the steps have a red asterisk (*). A red asterisk denotes that the step has incomplete data and requires a valid value. On the Summary page, verify the cell, node, and server on which the application is installed.

transition: You do not need to specify Java Naming and Directory Interface (JNDI) values for EJB bean, local home, remote home, or business interfaces of EJB 3.0 or later modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.0, EJB 3.1, Web 2.4, or Web 2.5 module, you do not need to specify JNDI values because the product resolves the targets automatically during run time. Even when you select **Generate default bindings**, the product does not generate default values for those JNDI values but it does generate default values for other bindings such as virtual host.

If you select **Generate default bindings**, the product generates bindings as follows:

- Enterprise bean (EJB) JNDI names are generated in the form *prefix/ejb-name*. The default prefix is *ejb*, but can be overridden. The *ejb-name* is as specified in the deployment descriptors `<ejb-name>` tag

or in its corresponding annotation for EJB 3.0 or later modules. The product does not generate default values for enterprise beans in an EJB 3.0 or later module because the run time provides container default values.

- EJB references are bound if an `<ejb-link>` is found. Otherwise, if a unique enterprise bean is found with a matching home (or local home) interface as the referenced bean, the reference is resolved automatically. The product does not generate default values for EJB reference in an EJB 3.0, EJB 3.1, Web 2.4, or Web 2.5 module because the run time provides container default values or automatically resolves the target references.
- Resource reference bindings are derived from the `<res-ref-name>` tag or its corresponding annotation for Java Platform, Enterprise Edition (Java EE) 5 or 6 modules. This action assumes that the `java:comp/env` name is the same as the resource global JNDI name.
- Connection factory bindings for EJB 2.0 and EJB 2.1 JAR files are generated based on the JNDI name and authorization information provided. This action results in default connection factory settings for each EJB 2.0 and EJB 2.1 JAR file in the application being installed. No bean-level connection factory bindings are generated.
- Data source bindings for EJB 1.1 JAR files are generated based on the JNDI name, data source user name and password options. This action results in default data source settings for each JAR file. No bean-level data source bindings are generated.
- For EJB 2.0 or later message-driven beans deployed as Java EE Connector Architecture (JCA) 1.5-compliant resources, the JNDI names corresponding to activationSpec instances are generated in the form `eis/MDB_ejb-name`. Message destination references are bound if a `<message-destination-link>` is found, then the JNDI name is set to `ejs/message-destination-linkName`. Otherwise, the JNDI name is set to `eis/message-destination-refName`.
- For EJB 2.0 or later message-driven beans deployed against listener ports, the listener ports are derived from the message-driven bean `<ejb-name>` tag with the string `Port` appended.
- For `.war` files, the virtual host is set as `default_host` unless otherwise specified.

The default strategy suffices for most applications or at least for most bindings in most applications. However, if you experience errors, complete the following actions:

- Control the global JNDI names of one or more EJB files.
- Control data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- Map resource references to global resource JNDI names that are different from the `java:comp/env` name.

In such cases, you can change the behavior with an XML document, which is a custom strategy. Use the **Specific bindings file** setting to specify a custom strategy and see the setting description in this help file for examples.

Override existing bindings

Specifies whether generated bindings are to replace existing bindings.

The default is to not override existing bindings. Select **Override existing bindings** to have generated bindings replace existing bindings.

Override existing bindings is similar to the `-defaultbinding.force` scripting option.

Specific bindings file

Specifies a bindings file that overrides the default binding.

Specific bindings file is similar to the `-defaultbinding.strategy.file` scripting option.

Change the behavior of the default binding with an XML document, which is a custom strategy. Custom strategies extend the default strategy so you only need to customize those areas where the default strategy is insufficient. Thus, you only need to describe how you want to change the bindings generated by the default strategy; you do not have to define bindings for the entire application.

Use the following examples to override various aspects of the default bindings generator:

Controlling an EJB JNDI name

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>HelloEjb</ejb-name>
          <jndi-name>com/acme/ejb/HelloHome</jndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Remember: Ensure that the setting for <ejb-name> matches the ejb-name entry in the EJB JAR deployment descriptor. Here the setting is <ejb-name>HelloEjb</ejb-name>.

Setting the connection factory binding for an EJB JAR file

```
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <connection-factory>
        <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
        <res-auth>Container</res-auth>
      </connection-factory>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Setting the connection factory binding for an EJB file

```
<?xml version="1.0">
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourCmp20</ejb-name>
          <connection-factory>
            <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
            <res-auth>PerConnFact</res-auth>
          </connection-factory>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Restriction: Ensure that the setting for <ejb-name> matches the ejb-name tag in the deployment descriptor. Here the setting is <ejb-name>YourCmp20</ejb-name>.

Setting the message destination reference JNDI for a specific enterprise bean

This example shows an XML extract in a custom strategy file for setting message-destination-refs for a specific enterprise bean.

```
<?xml version="1.0">
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb21.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourSession21</ejb-name>
          <message-destination-ref-bindings>
```

```

    <message-destination-ref-binding>
      <message-destination-ref-name>jdbc/MyDataSrc</message-destination-ref-name>
      <jndi-name>eis/somA0</jndi-name>
    </message-destination-ref-binding>
  </message-destination-ref-bindings>
</ejb-binding>
</ejb-bindings>
</ejb-jar-binding>
</module-bindings>
</dfldbndngs>

```

Restriction: Ensure that the setting for <ejb-name> matches the ejb-name tag in the deployment descriptor. Here the setting is <ejb-name>YourSession21</ejb-name>. Also ensure that the setting for <message-destination-ref-name> matches the message-destination-ref-name tag in the deployment descriptor. Here the setting is <message-destination-ref-name>jdbc/MyDataSrc</message-destination-ref-name>.

Overriding a resource reference binding from a WAR, EJB JAR file, or Java EE client JAR file

This example shows code for overriding a resource reference binding from a WAR file. Use similar code to override a resource reference binding from an enterprise bean (EJB) JAR file or a Java EE client JAR file.

```

<?xml version="1.0"?>
<!DOCTYPE dfldbndngs SYSTEM "dfldbndngs.dtd">
<dfldbndngs>
  <module-bindings>
    <war-binding>
      <jar-name>hello.war</jar-name>
      <resource-ref-bindings>
        <resource-ref-binding>
          <resource-ref-name>jdbc/MyDataSrc</resource-ref-name>
          <jndi-name>war/override/dataSource</jndi-name>
        </resource-ref-binding>
      </resource-ref-bindings>
    </war-binding>
  </module-bindings>
</dfldbndngs>

```

Restriction: Ensure that the setting for <resource-ref-name> matches the resource-ref tag in the deployment descriptor. In the previous example, the setting is <resource-ref-name>jdbc/MyDataSrc</resource-ref-name>.

Overriding the JNDI name for a message-driven bean deployed as a JCA 1.5-compliant resource

This example shows an XML extract in a custom strategy file for overriding the Java Message Service (JMS) activationSpec JNDI name for an EJB 2.0 or later message-driven bean deployed as a JCA 1.5-compliant resource.

```

<?xml version="1.0"?>
<!DOCTYPE dfldbndngs SYSTEM "dfldbndngs.dtd">
<dfldbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>YourEjbJar.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourMDB</ejb-name>
          <activationSpecJndi-name>activationSpecJNDI</activationSpecJndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfldbndngs>

```

Overriding the JMS listener port name for an EJB 2.0, 2.1, or 3.0 message-driven bean

This example shows an XML extract in a custom strategy file for overriding the JMS listener port name for an EJB 2.0 or later message-driven bean deployed against a listener port.

```

<?xml version="1.0"?>
<!DOCTYPE dfldbndngs SYSTEM "dfldbndngs.dtd">
<dfldbndngs>

```

```

<module-bindings>
  <ejb-jar-binding>
    <jar-name>YourEjbJar.jar</jar-name>
    <ejb-bindings>
      <ejb-binding>
        <ejb-name>YourMDB</ejb-name>
        <listener-port>yourMdbListPort</listener-port>
      </ejb-binding>
    </ejb-bindings>
  </ejb-jar-binding>
</module-bindings>
</df1tbindngs>

```

Overriding an EJB reference binding from an EJB JAR, WAR file, or EJB file

This example shows code for overriding an EJB reference binding from an EJB JAR file. Use similar code to override an EJB reference binding from a WAR file or an EJB file.

```

<?xml version="1.0"?>
<!DOCTYPE df1tbindngs SYSTEM "df1tbindngs.dtd">
<df1tbindngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>YourEjbJar.jar</jar-name>
      <ejb-ref-bindings>
        <ejb-ref-binding>
          <ejb-ref-name>YourEjb</ejb-ref-name>
          <jndi-name>YourEjb/JNDI</jndi-name>
        </ejb-ref-binding>
      </ejb-ref-bindings>
    </ejb-jar-binding>
  </module-bindings>
</df1tbindngs>

```

Specify unique prefix for beans

Specifies a string that the product applies to the beginning of generated enterprise bean JNDI names. The prefix must be unique within the cell or node.

The default is to not specify a unique prefix for beans.

Specify unique prefix for beans is similar to the scripting option `-defaultbinding.ejbjndi.prefix`.

Default bindings for EJB 1.1 CMP beans

Specifies the default data source JNDI name and other bindings for container-managed persistence (CMP) 1.1 beans.

The default is to not use default bindings for EJB 1.1 CMP beans.

If you select **Default bindings for EJB 1.1 CMP beans**, specify the JNDI name for the default data source to be used with the CMP 1.1 beans. Also specify the user name and password for this default data source.

Default bindings for EJB 1.1 CMP beans is similar to the scripting option `-defaultbinding.datasource.jndi`.

Default connection factory bindings

Specifies the default connection factory JNDI name.

The default is to not use default connection factory bindings. Select **Default connection factory bindings** to specify bindings for connection factories.

If you select **Default connection factory bindings**, specify the JNDI name for the default connection factory to be used. Also specify whether the resource authorization is for the application or container-wide.

Default connection factory bindings is similar to the scripting option `-defaultbinding.cf.jndi`.

Use default virtual host name for web and SIP modules

Specifies the virtual host for the web module (WAR file) or Session Initiation Protocol (SIP) module (SAR file).

The default is to not use default virtual host name for web or SIP modules. If you select **Use default virtual host name for web and SIP modules**, specify a default host name.

Use default virtual host name for Web and SIP modules is similar to the scripting option `-defaultbinding.virtual.host`.

Select installation options settings

Use this page to specify options for the installation of a Java Platform, Enterprise Edition (Java EE) application onto a WebSphere Application Server deployment target. Default values for the options are used if you do not specify a value. After application installation, you can specify values for many of these options from an enterprise application settings page.

To view this administrative console page, click **Applications > New application > New Enterprise Application** and then specify values as needed for your application on the Preparing for application installation pages.

The Select installation options page is the same for the application installation and update wizards.

Precompile JavaServer Pages files

Specify whether to precompile JavaServer Pages (JSP) files as a part of installation. The default is not to precompile JSP files.

For this option, install only onto a Version 8.5 deployment target.

If you select **Precompile JavaServer Pages files** and try installing your application onto an earlier deployment target such as Version 7, the installation is rejected. You can deploy applications to only those deployment targets that have same version as the product. If applications are targeted to servers that have an earlier version than the product, then you cannot deploy to those targets.

Information	Value
Data type	Boolean
Default	false

Directory to install application

Specifies the directory to which the enterprise archive (EAR) file will be installed.

By default, the EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

Setting options include the following:

- Do not specify a value and leave the field empty.

The default value is `${APP_INSTALL_ROOT}/cell_name`, where the `${APP_INSTALL_ROOT}` variable is *profile_root/installedApps*. A directory having the EAR file name of the application being installed is appended to `${APP_INSTALL_ROOT}/cell_name`. Thus, if you do not specify a directory, the EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

- Specify a directory.

If you specify a directory for **Directory to install application**, the application is installed in *specified_path/application_name.ear* directory. A directory having the EAR file name of the application being installed is appended to the path that you specify for **Directory to install application**. For

example, if you are installing C1ock.ear and specify C:/myapps on Windows computers, the application is installed in the myapps/C1ock.ear directory. The `${APP_INSTALL_ROOT}` variable is set to the specified path.

- Specify `${APP_INSTALL_ROOT}/${CELL}` for the initial installation of the application.

If you intend to export the application from one cell and later install the exported application on a different cell, specify the `${CELL}` variable for the initial installation of the application. For example, specify `${APP_INSTALL_ROOT}/${CELL}` for this setting. Exporting the application creates an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the cell name of the initial installation in the destination directory unless you specify the `${CELL}` variable. Specifying the `${CELL}` variable ensures that the destination directory has the current cell name, and not the original cell name.

Important: If an installation directory is not specified when an application is installed on a single-server configuration, the application is installed in `${APP_INSTALL_ROOT}/cell_name`. When the server is made a part of a multiple-server configuration (using the `addNode` utility), the cell name of the new configuration becomes the cell name of the deployment manager node. If the `-includeapps` option is used for the `addNode` utility, then the applications that are installed prior to the `addNode` operation still use the installation directory `${APP_INSTALL_ROOT}/cell_name`. However, an application that is installed after the server is added to the network configuration uses the default installation directory `${APP_INSTALL_ROOT}/network_cell_name`. To move the application to the `${APP_INSTALL_ROOT}/network_cell_name` location upon running the `addNode` operation, explicitly specify the installation directory as `${APP_INSTALL_ROOT}/${CELL}` during installation. In such a case, the application files can always be found under `${APP_INSTALL_ROOT}/current_cell_name`.

- If the application has been exported and you are installing the exported EAR file in a different cell or location, specify `${APP_INSTALL_ROOT}/cell_name/application_name.ear` if you did not specify `${APP_INSTALL_ROOT}/${CELL}` for the initial installation.

The exported EAR file is an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the value for **Directory to install application** that was used for the previous installation of the application. Unless you specify a different value for **Directory to install application** for this installation, the enhanced EAR file will be installed to the same directory as for the previous installation.

If you did not specify the `${CELL}` variable during the initial installation, the deployment configuration uses the cell name of the initial installation in the destination directory. If you are installing on a different cell, specify `${APP_INSTALL_ROOT}/cell_name/application_name.ear`, where `cell_name` is the name of the cell to which you want to install the enhanced EAR file. If you do not designate the current cell name, `cell_name` will be the original cell name even though you are installing the enhanced EAR file on a cell that has a different name.

- Specify an absolute path or a use pathmap variable.

You can specify an absolute path or use a pathmap variable such as `${MY_APPS}`. You can use a pathmap variable in any installation.

A pathmap variable is particularly needed when installing an application on a cluster with members on heterogeneous nodes because, in such cases, there might not be a single way to specify an absolute path. A WebSphere Application Server variable `${CELL}` that denotes the current cell name can also be in the pathmap variable; for example, `${MY_APP}/${CELL}`. You can define WebSphere Application Server variables on the WebSphere variables page, accessed by clicking **Environment > WebSphere variables**.

This **Directory to install application** field is the same as the **Location (full path)** setting on an Application binaries page.

Information	Value
Data type	String

Information	Value
Units	Full path name

Distribute application

Specifies whether the product expands application binaries in the installation location during installation and deletes application binaries during uninstallation. The default is to enable application distribution. Application binaries for installed applications are expanded to the directory specified.

On single-server products, the binaries are deleted when you uninstall and save changes to the configuration.

On multiple-server products, the binaries are deleted when you uninstall and save changes to the configuration and synchronize changes.

If you disable this option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application runs.

Note: If you disable this option and you do not copy and expand the application binaries to the nodes, a later saving of the configuration or manual synchronization does not move the application binaries to the nodes for you.

This **Distribute application** field is the same as the **Enable binary distribution, expansion and cleanup post uninstallation** setting on an Application binaries page.

Information	Value
Data type	Boolean
Default	true

Use binary configuration

Specifies whether the application server uses the binding, extensions, and deployment descriptors located with the application deployment document, the `deployment.xml` file (default), or those located in the enterprise archive (EAR) file. Select this setting for applications installed on Version 6.0 or later deployment targets only.

The default (false) is to use the binding, extensions, and deployment descriptors located in `deployment.xml`. To use the binding, extensions, and deployment descriptors located in the EAR file, enable this setting (true).

This **Use binary configuration** field is the same as the **Use configuration information in binary** setting on an Application binaries page.

Information	Value
Data type	Boolean
Default	false

Deploy enterprise beans

Specifies whether the EJBDeploy tool runs during application installation.

The tool generates code needed to run Enterprise JavaBeans (EJB) files. You must enable this setting in the following situations:

- The EAR file was assembled using an assembly tool such as Rational Application Developer and the EJBDeploy tool was not run during assembly.
- The EAR file was not assembled using an assembly tool such as Rational Application Developer.

- The EAR file was assembled using versions of the Application Assembly Tool (AAT) previous to Version 5.0.

If an EJB module is packaged in a web archive (WAR), you do not need to enable this setting.

The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.x modules.

For this option, install only onto a Version 8.5 deployment target.

If you select **Deploy enterprise beans** and try installing your application onto an earlier deployment target such as Version 7, the installation is rejected. You can deploy applications to only those targets that have same WebSphere version as the product. If applications are targeted to servers that have an earlier version than the product, then you cannot deploy to those targets.

Also, if you select **Deploy enterprise beans** and specify a database type on the **Provide options to perform the EJB Deploy** page, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type. To enable backend IDs for individual EJB modules, set the database type to "" (null) on the **Provide options to perform the EJB Deploy** page.

Enabling this setting might cause the installation program to run for several minutes.

Information	Value
Data type	Boolean
Default	true (false for EJB 3.0 modules)

Application name

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Table 22. Characters that you cannot use in a name. The product does not support these characters in a name.

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

This **Application name** field is the same as the **Name** setting on an Enterprise application settings page.

Information	Value
Data type	String

Create MBeans for resources

Specifies whether to create MBeans for resources such as servlets or JSP files within an application when the application starts. The default is to create MBeans.

This field is the same as the **Create MBeans for resources** setting on a Startup behavior page.

Information	Value
Data type	Boolean
Default	true

Override class reloading settings for web and EJB modules

Specifies whether the product run time detects changes to application classes when the application is running. If this setting is enabled and if application classes are changed, then the application is stopped and restarted to reload updated classes.

The default is not to enable class reloading.

This field is the same as the **Override class reloading settings for web and EJB modules** setting on a Class loading and update detection page.

Information	Value
Data type	Boolean
Default	false

Reload interval in seconds

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

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

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

This **Reload interval in seconds** field is the same as the **Polling interval for updated files** setting on a Class loading and update detection page.

Information	Value
Data type	Integer
Units	Seconds
Default	3

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Deploy web services

Specifies whether the web services deploy tool `wsdeploy` runs during application installation.

The tool generates code needed to run applications using web services. The default is not to run the `wsdeploy` tool. You must enable this setting if the EAR file contains modules using web services and has not previously had the `wsdeploy` tool run on it, either from the **Deploy** menu choice of an assembly tool or from a command line.

For this option, install only onto a Version 8.5 deployment target.

If you select **Deploy web services** and try installing your application onto an earlier deployment target, the installation is rejected. You can deploy applications to only those targets that have same version as the product. If applications are targeted to servers that have an earlier version than the product, then you cannot deploy to those targets.

Information	Value
Data type	Boolean
Default	false

Validate input off/warn/fail

Specifies whether the product examines the application references specified during application installation or updating and, if validation is enabled, warns you of incorrect references or fails the operation.

An application typically refers to resources using data sources for container managed persistence (CMP) beans or using resource references or resource environment references defined in deployment descriptors. The validation checks whether the resource referred to by the application is defined in the scope of the deployment target of that application.

Select **off** for no resource validation, **warn** for warning messages about incorrect resource references, or **fail** to stop operations that fail as a result of incorrect resource references.

This **Validate input off/warn/fail** field is the same as the **Application reference validation** setting on an Enterprise application settings page.

Information	Value
Data type	String
Default	warn

Process embedded configuration

Specifies whether the embedded configuration should be processed. An embedded configuration consists of files such as `resource.xml` and `variables.xml`. When selected or `true`, the embedded configuration is loaded to the application scope from the `.ear` file. If the `.ear` file does not contain an embedded configuration, the default is `false`. If the `.ear` file contains an embedded configuration, the default is `true`.

This setting affects installation of enhanced EAR files. An enhanced EAR file results when you export an installed application.

When `false`, an enhanced EAR file is installed like any other application and the product ignores its embedded configuration.

If you exported the application from a cell other than the current cell and did not specify the `$(CELL)` variable for **Directory to install application** when first installing the application, deselect this setting (`false`) to expand the enhanced EAR file in the `profile_root/installedApps/current_cell_name` directory. Otherwise, if this setting is selected (`true`), the enhanced EAR file is expanded in the `profile_root/installedApps/original_cell_name` directory, where `original_cell_name` is the cell on which

the application was first installed. If you specified the \$(CELL) variable for **Directory to install application** when you first installed the application, installation expands the enhanced EAR file in the `profile_root/installedApps/current_cell_name` directory.

Information	Value
Data type	Boolean
Default	false (deselected)

File permission

Specifies access permissions for application binaries for installed applications that are expanded to the directory specified.

The **Distribute application** option must be enabled to specify file permissions.

You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the multiple-selection list. List selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the list. Selecting multiple options combines the file permission strings.

Table 23. File permission string sets for list options. Select a list option or specify a file permission string in the text field.

Multiple-selection list option	File permission string set
Allow all files to be read but not written to	.*=755
Allow executables to execute	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Allow HTML and image files to be read by everyone	.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755

Instead of using the multiple-selection list to specify file permissions, you can specify a file permission string in the text field. File permissions use a string that has the following format:

`file_name_pattern=permission#file_name_pattern=permission`

where `file_name_pattern` is a regular expression file name filter (for example, `.*\.jsp` for all JSP files), `permission` provides the file access control lists (ACLs), and `#` is the separator between multiple entries of `file_name_pattern` and `permission`. If `#` is a character in a `file_name_pattern` string, use `\#` instead.

If multiple file name patterns and file permissions in the string match a uniform resource identifier (URI) within the application, then the product uses the most stringent applicable file permission for the file. For example, if the file permission string is `.*\.jsp=775#a.*\.jsp=754`, then the `abc.jsp` file has file permission 754.

best-practices: Using regular expressions for file matching pattern compares an entire string URI against the specified file permission pattern. You must provide more precise matching patterns using regular expressions as defined by Java programming API. For example, suppose the following directory and file URIs are processed during a file permission operation:

Table 24. Example URIs for file permission operations. Results are shown following this table.

Number	Example URL
1	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war
2	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
3	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/META-INF/MANIFEST.MF
4	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/WEB-INF/classes/MyClass.class

Table 24. Example URIs for file permission operations (continued). Results are shown following this table.

Number	Example URL
5	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/mydir/MyClass2.class
6	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF

The file pattern matching results are:

- MyWarModule.war does not match any of the URIs
- .*MyWarModule.war.* matches all URIs
- .*MyWarModule.war\$ matches only URI 1
- .*\\.jsp=755 matches only URI 2
- .*META-INF.* matches URIs 3 and 6
- .*MyWarModule.war/.*/.*\\.class matches URIs 4 and 5

If you specify a directory name pattern for **File permissions**, then the directory permission is set based on the value specified. Otherwise, the **File permissions** value set on the directory is the same as its parent. For example, suppose you have the following file and directory structure:

```
/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
```

and you specify the following file pattern string:

```
.*MyApp.ear$=755#.*\\.jsp=644
```

The file pattern matching results are:

- Directory MyApp.ear is set to 755
- Directory MyWarModule.war is set to 755
- Directory MyWarModule.war is set to 755

best-practices: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

Access permissions specified here are at the application level. You can also specify access permissions for application binaries in the node-level configuration. The node-level file permissions specify the maximum (most lenient) permissions that can be given to application binaries. Access permissions specified here at application level can only be the same as or more restrictive than those specified at the node level.

This setting is the same as the **File permissions** field on the Application binaries page.

Information	Value
Data type	String

Application build identifier

Specifies an uneditable string that identifies the build version of the application.

This **Application build identifier** field is the same as the **Application build level** field on the Application binaries page.

Information	Value
Data type	String

Allow dispatching includes to remote resources

Specifies whether an application can dispatch includes to resources across web modules that are in different Java virtual machines in a managed node environment through the standard request dispatcher mechanism.

This field is the same as the **Allow dispatching includes to remote resources** field on the Remote request dispatch properties page.

Information	Value
Data type	Boolean
Default	false

Allow servicing includes from remote resources

Specifies whether an enterprise application can service an include request from an application.

This field is the same as the **Allow servicing includes from remote resources** field on the Remote request dispatch properties page.

Information	Value
Data type	Boolean
Default	false

Business-level application name

Specifies whether the product creates a new business-level application with the enterprise application that you are installing or makes the enterprise application a composition unit of an existing business-level application.

The default is to create a new business-level application with a setting value of `WebSphere:blaname=Anyasset,blaedition=BASE`. When you select to create a new business-level application from the drop-down list, the product creates a business-level application that has the same name as your enterprise application. If a business-level application with the name of your enterprise application exists already, the product does not create a new business-level application; it adds your enterprise application as a composition unit to that existing business-level application.

If you need to use the Shared library relationship and mapping settings page to specify dependency relationships on existing shared libraries in the business-level application, select the business-level application name from the drop-down list. No shared libraries are shown in the page if you choose to create a new business-level application and a business-level application with the default name exists already.

To add your enterprise application to an existing business-level application, select an existing business-level application from the drop-down list. The product makes your enterprise application a composition unit of the existing business-level application.

Information	Value
Data type	String
Default	Create a new business-level application that has the same name as the enterprise application that you are installing.

`WebSphere:blaname=Anyasset,blaedition=BASE`

Asynchronous request dispatch type

Specifies whether web modules can dispatch requests concurrently on separate threads and, if so, whether the server or client dispatches the requests. Concurrent dispatching can improve servlet response time.

If operations are dependant on each other, do not enable asynchronous request dispatching. Select **Disabled**. Concurrent dispatching might result in errors when operations are dependant.

Select **Server side** to enable the server to dispatch requests concurrently. Select **Client side** to enable the client to dispatch requests concurrently.

Information	Value
Data type	String
Default	Disabled

Allow EJB reference targets to resolve automatically

Specifies whether the product assigns default JNDI values for or automatically resolves incomplete EJB reference targets.

Select this option to enable EJB reference targets to resolve automatically if the references are from EJB 2.1 or earlier modules or from Web 2.3 or earlier modules. If you enable this option, the runtime container provides a default value or automatically resolves the EJB reference for any EJB reference that does not have a binding.

If you selected **Generate default bindings** on the Preparing for application installation page, then you do not need to select this option. The product generates default values.

If you select **Allow EJB reference targets to resolve automatically**, all modules in the application must share one deployment target. If you select this option and all of the application modules do not share a common server, after you click **Finish** on the Summary page, the product displays a warning message and does not install the application. You must deselect this setting before you click **Finish** to install the application.

Information	Value
Data type	Boolean
Default	false

Deploy client modules

Specifies whether to deploy client modules.

Select this option (set to `true`) if the file to deploy has one or more client modules and you want to configure environment entries for the client modules. Also select this option to configure resources such as EJB references, resource references, resource environment references, or message destination references. Selecting this option enables you to view the Map environment entries for client modules page. If you are deploying the client modules to a federated node of a deployment manager (**Federated**) or to an application server (**Server Deployed**), select this option and set **Client deployment mode** to the appropriate value for the deployment target, **Federated** or **Server Deployed**.

If you select this option, install the client modules only onto a Version 8.0 or later deployment target.

Information	Value
Data type	Boolean
Default	false

Client deployment mode

Specifies whether to deploy client modules to an isolated deployment target (**Isolated**), a federated node of a deployment manager (**Federated**), or an application server (**Server Deployed**).

The choice of client deployment mode affects how `java:` lookups are handled. All Java URL name spaces (global, application, module, and component) are local in isolated client processes. The name spaces reside on a server in federated and server deployed client processes. The server or cluster chosen as a target for a client module determines where those name spaces are created. All `java:` lookups for

federated or server deployed client modules are directed to the target server or cluster. The client module does not actually run in the target server or cluster. Multiple instances of the same client module will all share the component name space in the **Federated** and **Server Deployed** modes. Choosing the **Federated** mode is simply a declaration of intent to launch the client module using Java Network Launching Protocol (JNLP), but the Java Naming and Directory Interface (JNDI) mechanics of federated and server deployed modes are the same.

Information	Value
Data type	String
Default	Isolated

Validate schema

Specifies whether to validate the deployment descriptors against published Java EE deployment descriptor schemas. When this option is selected, the product analyzes each deployment descriptor to determine the Java EE specification version for the deployment descriptor, selects the appropriate schema, and then checks the deployment descriptor against the Java EE deployment descriptor schema. Validation errors result in error messages.

A Java EE deployment descriptor schema is also known as a *DTD*.

If you select this option, install your application or module only onto a Version 8.0 or later deployment target.

Information	Value
Data type	Boolean
Default	false

Manage modules settings

Use this page to specify deployment targets where you want to install the modules that are contained in your application. Modules can be installed on the same deployment target or dispersed among several deployment targets.

On single-server products, a deployment target can be an application server or web server.

On multiple-server products, a deployment target can be an application server, cluster of application servers, or web server.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Manage modules**. This page is similar to the Map modules to servers page on the application installation and update wizards.

On this page, each **Module** must map to one or more targets, identified under **Server**. To change a mapping:

1. In the list of mappings, select each module that you want mapped to the same target or targets.
2. If your security configuration supports multiple domains, select the domain that has the desired clusters or servers or keep the **All domains** default.
3. From the **Clusters and servers** list, select one or more targets. Select only appropriate deployment targets for a module. You cannot install modules that use WebSphere Application Server Version 8.x features on a Version 7.x or 6.x target server. Similarly, you cannot install modules that use Version 7.x features on a Version 6.x target server.

Use the Ctrl key to select multiple targets. For example, to have a web server serve your application, press the Ctrl key and then select an application server and the web server together. The product generates the plug-in configuration file, `plugin-cfg.xml`, for that web server based on the applications which are routed through it.

4. Click **Apply**.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

If you accessed this Manage modules page from a console enterprise application page for an already installed application, you can also use this page to view and manage modules in your application.

To view the values specified for a module configuration, click the module name in the list. The displayed module settings page shows the values specified. On the settings page, you can change existing configuration values and link to additional console pages that assist you in configuring the module.

To manage a module, select the module name in the list and click a button:

Table 25. Button descriptions. Use the buttons to manage modules.

Button	Resulting action
Remove	Removes the selected module from the deployed application. The module is deleted from the application in the configuration repository and also from all of the nodes where the application is installed and running or expected to run. On multiple-server products, if the application is running on a node when the module file is deleted from the node as a result of configuration synchronization then the product stops the application, deletes the module file from the file system of the node, and restarts the application.
Update	Opens a wizard that helps you update modules in an application. If a module has the same URI as a module already existing in the application, the new module replaces the existing module. If the new module does not exist in the application, it is added to the deployed application. On multiple-server products, if the application is running on a node when the module file is updated on the node as a result of configuration synchronization then the product stops the application, updates the module file on the file system of the node, and restarts the application. If the application is running on a node when the module file is added as a result of configuration synchronization then the product starts the newly added module without stopping and restarting the running application.
Remove File	Deletes a file from a module of a deployed application. On multiple-server products, the file is also deleted from all the nodes where the module is installed after configuration is synchronized with nodes. If the application is running on a node when the module file is deleted from the node as a result of configuration synchronization then the product stops the application, deletes the module file from the file system of the node, and restarts the application.
Export File	Accesses the Export a file from an application page, which you use to export a file of an enterprise application or module to a location of your choice. If the browser does not prompt for a location to store the file, click File > Save as and specify a location to save the file that is shown in the browser.

Display clusters and servers in the following domain

Lists the domains that your security configuration supports. This setting displays only on multiple-server products with a security configuration that supports multiple domains.

From this list, select the domain that has the clusters or servers on which you want to deploy an application or module. To see all available deployment targets, select **All Domains**.

Selecting a domain causes the **Clusters and servers** list to show only the deployment targets that are configured in the domain. You cannot deploy the modules in an application across deployment targets that belong to different security domains.

Clusters and servers

Lists the names of available deployment targets. This list is the same for every application that is installed in the cell.

From this list, select only appropriate deployment targets for a module. You must install an application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) archive (SAR), web module, or client module on a Version 8.x target under any of the following conditions:

- The module supports Java Platform, Enterprise Edition (Java EE) 6.
- The module calls an 8.x runtime application programming interface (API).
- The module uses an 8.x product feature.

You must install an application, EJB, SAR, or web module on a Version 8.x or 7.x target under any of the following conditions:

- The module supports Java EE 5.
- The module calls a 7.x runtime API.
- The module uses a 7.x product feature.

If a module supports J2EE 1.4, then you must install the module on a Version 6.x, 7.x or 8.x deployment target. Modules that call a 6.1.x API or use a 6.1.x feature can be installed on a 6.1.x, 7.x or 8.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 6.1.x deployment target that has been enabled with that feature pack or on a 7.x or 8.x deployment target.

You can install an application or module developed for a Version 5.x product on any deployment target.

Module

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

URI

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

Module type

Specifies the type of module, for example, a web module or EJB module.

This setting is shown on the Manage modules page accessed from a console enterprise application page.

Server

Specifies the name of each deployment target to which the module currently is mapped.

To change the deployment targets for a module, select one or more targets from the **Clusters and servers** list and click **Apply**. The new mapping replaces the previous mapping.

Client module settings

Use this page to configure a deployed client module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Manage modules > *client_module_name***. This page is viewable only if the selected application contains a client module and the client deployment mode is a value other than `isolated`.

URI

Specifies the location of the client module relative to the root of the application.

Alternate deployment descriptor

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

Client module property settings

Use this page to configure the deployment mode of a deployed client module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Client module deployment mode**. This page is viewable only if the selected application contains a client module.

Client module deployment mode

Specifies whether to deploy client modules to an isolated deployment target (**Isolated**), a federated node of a deployment manager (**Federated**), or an application server (**Server Deployed**).

Information	Value
Data type	String
Default	Isolated

Provide options to compile JavaServer Pages settings

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

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

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

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The ibm-webservices-ext.xml, ibm-webservices-bnd.xml, ibm-webservicesclient-bnd.xml, ibm-webservicesclient-ext.xml, and ibm-portlet-ext.xml files continue to use the .xmi file extensions.

Web module

Specifies the name of a module within the application.

URI

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

JSP class path

Specifies a temporary class path for the JSP compiler to use when compiling JSP files during application installation. This class path is not saved when the application installation is complete and is not used when

the application is running. This class path is used only to identify resources outside of the application which are necessary for JSP compilation and which will be identified by other means (such as shared libraries) after the application is installed. In network deployment configurations, this class path is specific to the deployment manager machine.

To specify that multiple web modules use the same class path:

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

Use full package names

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

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

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

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

JDK source level

Specifies the source level at which the Java compiler compiles JSP Java sources. Valid values are 13, 14, and 15. The default value is 13 for pre-Java EE 5 web modules, which specifies source level 1.3 and 15 for Java EE 5 and later web modules.

Disable JSP runtime compilation

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

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

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

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

EJB JNDI names for beans

Use this page to view and modify the Java Naming and Directory Interface (JNDI) names of non-message-driven enterprise beans in your application or module.

If your application uses Enterprise JavaBeans (EJB) 2.1 and earlier modules, on the Provide JNDI names for beans panel, specify a JNDI name for each enterprise bean in every EJB 2.1 and earlier module. You must specify a JNDI name for every EJB 2.1 and earlier enterprise bean defined in the application. For example, for the EJB module `MyBean.jar`, specify `MyBean`.

The JNDI name for an EJB module can be used for both EJB 3.x modules and pre-EJB 3.0 modules. For a pre-EJB 3.0 module, you need to provide a JNDI name for the bean. For an EJB 3.x module, you have three options

- Provide no JNDI names at all
- Select the radio button to provide a JNDI name for the bean, or
- Select the radio button to provide local or remote home JNDI names.

If no JNDI name is provided, the run time provides a default value. If JNDI name for the bean is provided, you cannot provide any JNDI name for business interface in the Provide JNDI names for business interfaces panel.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > application > EJB JNDI names**.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Module

Specifies the name of the Enterprise JavaBeans module used by your application.

Bean

Specifies the name of an enterprise bean that is contained by the module.

URI

The Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.

Target Resource JNDI name

Specifies the Java Naming and Directory Interface (JNDI) name of the enterprise bean.

This is a data entry field. To modify the JNDI name bound to this bean, type the new name in this field, then select **OK**.

Information	Value
Data type	String

Bind EJB business settings

Use this administrative console page to specify Java Naming and Directory (JNDI) name bindings for each enterprise bean with a business interface in an EJB module. Each enterprise bean with a business interface in an EJB module must be bound to a JNDI name. For any business interface that does not provide a JNDI name, or if its bean does not provide a JNDI name, a default binding name is provided. If its bean provides a JNDI name, the default JNDI name for the business interface is provided on top of its bean JNDI name by appending the package-qualified class name of the interface.

If you specify the JNDI name for a bean in the Provide JNDI names for beans page, do not specify any business interface JNDI name in this page for the same bean. If you do not specify the JNDI name for a bean in the Provide JNDI names for beans page, you can optionally specify a business interface JNDI name. If you do not specify a business interface JNDI name, the run time provides a container default.

To view this page in the administrative console, click **Applications > Application Types > WebSphere enterprise applications > application_name > Bind EJB business**.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Module

Specifies the EJB module that contains the enterprise beans that bind to the JNDI name.

Bean

Specifies the enterprise bean that binds to the JNDI name.

URI

The Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.

Business Interface

Specifies the enterprise bean business interface in an EJB module.

For a no-interface view, the business interface value is an empty string ("").

JNDI Name

Specifies the JNDI name associated with the enterprise bean business interface in an EJB module.

Map default data sources for modules containing 1.x entity beans

Use this page to set the default data source mapping for EJB modules that contain 1.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 1.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Map default data sources for modules containing 1.x entity beans**.

Guidelines for using this administrative console page:

- The page displays a table that depicts the EJB modules in your application that contain 1.x CMP beans.
- Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target of the EJB module *only* if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.
- To set your default data source mappings:
 1. Select a row. Be aware that if you check multiple rows on this page, the data source mapping target that you select in step 2 applies to all of those EJB modules.
 2. Click **Browse** to select a data source from the new page that is displayed, the Available Resources page. The Available Resources page shows all data sources that are available mapping targets for your EJB modules.
 3. Click **Apply**. The console displays the 1.x entity bean data sources page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
 4. *Before* you click **OK** to save your new configuration, set the security parameters for the data source. Use the following steps.
- To specify security settings for the default data source:
 1. Select a row. Be aware that if you check multiple rows on this page, the security settings that you select later apply to all of those data sources.
 2. Type in a user name and password that comprise the authentication alias for signing on to the data source. If these entries are not listed in the application Java Platform, Enterprise Edition (Java EE)

Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the information center topic on managing Java EE Connector Architecture authentication data entries for more information.

3. Click **Apply** that immediately follows the user name and password input fields.
- Repeat all of the previous steps as necessary.
 - Click **OK** to save your work.

Select

Select the check boxes of the rows that you want to edit.

EJB Module

The name of the module that contains the 1.x enterprise beans.

URI

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

JNDI name

The Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Information	Value
Data type	String

User name

The user name and password that comprise the authentication alias for securing the data source.

EJB references

Use this page to view and modify the Enterprise JavaBeans (EJB) references to the enterprise beans. References are logical names used to locate external resources for enterprise applications. References are defined in the application's deployment descriptor file. At deployment, the references are bound to the physical location (global Java Naming and Directory Interface (JNDI) name) of the resource in the target operational environment.

If your application defines EJB references, for **Map EJB references to beans**, specify JNDI names for enterprise beans that represent the logical names that are specified in EJB references. Each EJB reference defined in the application must be bound to an EJB file before clicking Finish in the Summary panel.

If the EJB reference is from an EJB 3.x, Web 2.4, Web 2.5, or Client 5.0 module, the JNDI name is optional. If the **Allow EJB reference targets to resolved automatically** option is enabled, the JNDI name is optional for all modules. The runtime provides a container default or automatically resolves the EJB reference if a binding is not provided.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > application_name > EJB references**.

Values are displayed for Lookup name and EJB Link if they are configured in the application. Only one of these values is allowed. If both are set, the value must be overridden by a target resource JNDI name.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Module

Specifies the name of the Enterprise JavaBeans module used by your application.

Bean

Specifies the name of an enterprise bean that is contained by the module.

URI

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

Resource Reference

Specifies the name of the EJB reference that is used in the enterprise bean, if applicable, and declared in the deployment descriptor of the application module.

Class

Specifies the name of a Java class associated with this enterprise bean.

Target Resource JNDI Name

Specifies the JNDI name of the enterprise bean.

This is a data entry field. To modify the JNDI name bound to this bean, type the new name in this field, then select **OK**.

Information

Data type

Value

String

Resource references

Use this page to designate how the resource references of application modules map to the actual resources that are configured for the application.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Resource references**.

You can also view this page during the **Map resource references to resources** step when you install an application.

- If your application uses any of the following resource types, you can set or reset their mapping configurations:
 - Default messaging JMS queues destinations
 - Default messaging JMS topic destinations
 - Data source
 - Generic JMS connection factory
 - Mail session
 - J2C connection factory
 - JMS queue connection factory for the JMS provider of WebSphere MQ
 - JMS queue destination for WebSphere MQ
 - JMS topic connection factory for WebSphere MQ
 - JMS topic destination for WebSphere MQ
 - Unified JMS connection factory for WebSphere MQ
 - URL configuration
- The page is composed of sections that correspond to each applicable resource type. Each section heading is the class name for the resource. If your application contains only one applicable resource type, you see only one section.

- Each section contains a table. Each table row depicts a resource reference within a specific module of your application.
- The rows contain the JNDI names of resource mapping targets for your references *only* if you bound them together during application assembly. You can modify those bindings on this administrative console page.
- To set your mappings:
 1. Select a row. If you want to apply the same mapping to multiple rows, complete the steps in the section, Set multiple JNDI names.
 2. Click **Browse** to view a new page listing of all resources that are available mapping targets for your application references.
 3. Select a resource and click **Apply**. The console displays the Resource references page again. The JNDI name of the selected resource mapping displays in the Target Resource JNDI Name field.
 4. Repeat the previous steps as necessary.
 5. If you are editing the resource references of an existing enterprise application, click **OK**. You now return to the general configuration page for your enterprise application. If you are installing the application and have completed the **Map resource references to resources** step, continue to the next step.
- **For data sources and connection factories:** Sections for these resource types contain an additional set of steps for modifying your security settings. Use the last column in the displayed table to view the authorization type for each resource configuration per application module. You can modify the corresponding authentication method only if the authorization type is container. Container-managed authorization indicates that the product performs signon to the resource rather than the enterprise bean code. The reconfiguring process differs slightly for each authentication method option:
 - When you want to assign no authentication method to a resource:
 1. Determine which resource configurations to designate with no authentication method.
 2. Select the appropriate table rows.
 3. Click **Modify Resource Authentication Method** and select **None** from the authentication method options that are displayed above the table.
 4. Click **Apply**.
 - When you want to assign the WebSphere Application Server DefaultPrincipalMapping login configuration to a resource:
 1. You must apply this option to each resource individually if you want to designate different authentication data aliases. See the topic, J2EE connector security, for more information about the default mapping configuration.
 2. Select the appropriate table rows.
 3. Click **Modify Resource Authentication Method** and select **Use default method** from the list of authentication method options that are displayed above the table.
 4. Select an authentication data entry or alias from the list.
 5. Click **Apply**.
 - When you want to assign a trusted context to a resource:
 1. You must have a data source that is running at least DB2 Version 9.1 for z/OS[®], and the data source must have trusted context enabled.
 2. You must have a data source server that is running at least DB2 Version 9.1 for z/OS, and the data source must have trusted context enabled.
 3. Select the appropriate table rows that have trusted context enabled.
 4. Click **Modify Resource Authentication Method** and select **Use trusted connections** from the authentication method options that are displayed above the table.
 5. Select an authentication alias from the list that matches an alias that is already defined in the DB2 data source. If you do not have an alias defined that is suitable, you must define a new alias.

6. Click **Apply**.
 7. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.
- When you want to assign a custom Java Authentication and Authorization Service (JAAS) login configuration to a resource:
1. See the topic, J2EE connector security, for more information about custom JAAS login configurations.
 2. Select the appropriate table row.
 3. Click **Modify Resource Authentication Method** and select **Use custom login configuration** from the authentication method options that are displayed above the table.
 4. Select an application login configuration from the list.
 5. Click **Apply**.
 6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Set multiple JNDI names

Use this option to set the same JNDI name on multiple resources with one operation.

Click **Set multiple JNDI names** to display a menu of JNDI names. If you make a selection from this list, it is applied to the **Target Resource JNDI Name** field of all the selected rows of the table.

Modify Resource Authentication Method

Use this panel to toggle the display of a panel above the table rows.

This use of this panel is described in the **For data sources and connection factories** section.

Extended Properties

Use this panel to set additional properties on the selected resource.

Select a single table row and click **Extended Properties** to set additional properties on the selected resource. For more details on using this function, see the documentation on extending DB2 data source definitions at the application level.

Select

Select the check boxes of the rows that you want to edit.

Module

The name of a module in the application.

Bean

The name of an enterprise bean that is contained by the module.

URI

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

Resource Reference

The name of a resource reference that is used in the enterprise bean, if applicable, and is declared in the deployment descriptor of the application module.

Target Resource JNDI name

The Java Naming and Directory Interface (JNDI) name of the resource that is the mapping target of the resource reference.

Information	Value
Data type	String

Login configuration

This column applies to data sources and connection factories only and refers to the authorization type and the authentication method for securing the resource.

Virtual hosts settings

Use this page to specify virtual hosts for web modules contained in your application. Web modules can be installed on the same virtual host or dispersed among several virtual hosts.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Virtual hosts**. This page is the same as the **Map virtual hosts for web modules** page on the application installation and update wizards.

On this page, each web module must map to a previously defined virtual host, identified under **Virtual host**. You can see information on previously defined virtual hosts by clicking **Environment > Virtual hosts** in the administrative console. Virtual hosts enable you to associate a unique port with a module or application. The aliases of a virtual host identify the port numbers defined for that virtual host. A port number specified in a virtual host alias is used in the URL that is used to access artifacts such as servlets and JavaServer Pages (JSP) files in a web module. For example, the alias `myhost:8080` is the `host_name:port_number` portion of the URL `http://myhost:8080/servlet/snoop`.

The default virtual host setting usually is `default_host`, which provides several port numbers through its aliases:

80	An internal, insecure port used when no port number is specified
9080	An internal port
9443	An external, secure port

Unless you want to isolate your web module from other modules or resources on the same node (physical machine), `default_host` is a suitable virtual host for your web module.

In addition to `default_host`, the product provides `admin_host`, which is the virtual host for the administrative console system application. `admin_host` is on port 9060. Its secure port is 9043. Do not select `admin_host` unless the web module relates to system administration.

To change a mapping:

1. In the list of mappings, select the **Select** check box beside each web module that you want mapped to a particular virtual host.
2. From the **Virtual host** drop-down list, select the desired virtual host. If you selected more than one virtual host in step 1:
 - a. Expand **Apply Multiple Mappings**.
 - b. Select the desired virtual host from the **Virtual Host** drop-down list.
 - c. Click **Apply**.
3. Click **OK**.

Attention: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Web module

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

Virtual host

Specifies the name of the virtual host to which the Web module is currently mapped.

Expanding the drop-down list displays a list of previously defined virtual hosts. To change a mapping, select a different virtual host from the list.

Do not specify the same virtual host for different web modules that have the same context root and are deployed on targets belonging to the same node even if the web modules are contained in different applications. Specifying the same virtual host causes a validation error.

Security role to user or group mapping

Use this page to specify the users and groups that are mapped to the security roles that are used with the enterprise application.

To view this administrative console page, click **Applications > Application types > WebSphere enterprise applications > application_name**. Under Detail Properties, click **Security role to user/group mapping**.

Table 26. User and group mapping. User and group mapping.

Button	Resulting action
Map Users	<p>Lists the users that are mapped to the specified role within this application.</p> <p>If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm</p>
Map Groups	<p>Lists the groups that are mapped to this specified role within this application.</p> <p>If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm</p>
Map Special Subjects	<p>This choice appears if multiple realms are being used. It enables you to map any of the following Special Subjects to a selected role:</p> <ul style="list-style-type: none">• All authenticated in application realm: All authenticated users that are in the applications realm, which specifies whether to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role. This selection also applies to all authenticated users regardless of the realm.• Everyone: map everyone to the selected role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.• None: Do not map anyone to the selected role <p>Attention:</p> <ul style="list-style-type: none">• If the secured realm cannot be reached, the left list is replaced with 3 text fields (that is, name, realm, and uid). You can add the user when the secured realm is not available. <p>It is not possible to map two subjects to the same role in this release of WebSphere Application Server.</p>

Role

Lists the specific capabilities to a user. Role privileges give users and groups permission to run as specified.

For example, you might map the user Joe to the administrator role, which enables user Joe to perform all of the tasks associated with the administrator role.

The authorization policy is only enforced when global security is enabled.

Mapped users

Lists the users that are mapped to the specified role within this application.

Special subjects

Lists which special subjects are mapped to the security role when an application uses multiple realms.

Mapped groups

Lists the groups that are mapped to this specified role within this application.

JASPI authentication enablement for applications

Use this page to enable or disable Java Authentication SPI (JASPI) authentication for an application or web module, and to specify the name of a JASPI authentication provider to be used for authenticating messages for the application or web module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications**. Select an application, and under Detail Properties, select **JASPI provider**.

Select JASPI provider

Select to indicate the web modules in the application that you wish to specify or to override the default JASPI authentication settings for.

Select one of the JASPI provider names to choose a provider to use to perform authentication of web requests for the selected Web module or the application.

To specify how JASPI authentication is performed for the selected web module or the application, choose one of the following:

Do not use JASPI

Select to disable JASPI authentication for the selected web module or for the application.

Inherit JASPI provider

Select to inherit the JASPI authentication settings from default values in the cell or domain security configuration, as appropriate.

When Inherit JASPI provider is selected for a web module, JASPI authentication settings for the selected module are the settings that are specified for the application.

When Inherit JASPI provider is selected for the application, JASPI authentication settings are the settings that are specified in the appropriate cell or domain security configuration.

Provider name

When a specific provider name is selected, that provider is used to perform authentication of web requests for the selected application or web module.

If JASPI authentication is enabled, and a specific provider name is not specified, then the default provider name is used. For more information, read about configuring a new JASPI authentication provider using the administrative console.

If JASPI authentication is disabled, or if no default provider is selected, JASPI authentication is not performed. Web authentication is then performed according to another authentication mechanism as selected in the cell or domain security configuration.

User RunAs collection

Use this page to map a specified user identity and password to a RunAs role. This panel enables you to specify application-specific privileges for individual users to run specific tasks using another user identity.

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

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Detail properties, click **User runAs roles**.

The enterprise beans that you install contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

Username

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role. After you specify the user name and password for the user and select a RunAs role, click **Apply**.

Note:

The use of the username field is dependent on whether system authorization facility (SAF) delegation is enabled or disabled.

- SAF delegation is enabled. The username field is NOT used.
- SAF delegation is disabled. The username field is used.

Information

Data type:

Value

String

Password

Specifies the password for the RunAs user.

Note:

The use of the password field is dependent on whether system authorization facility (SAF) delegation is enabled or disabled.

- SAF delegation is enabled. The password field is NOT used.
- SAF delegation is disabled. The password field is used.

Information

Data type:

Value

String

Role

Maps specific capabilities to a user.

The authorization policy is only enforced when global security is enabled.

Ensure all unprotected 1.x methods have the correct level of protection

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 1.x methods have the correct level of protection before you map users to roles.

This administrative console page is displayed during the application deployment process. To access the administrative console page, click **Application > New application > New Enterprise Application**. The page is displayed as **Ensure all unprotected 1.x methods have the correct level of protection** in the application deployment steps. On this administrative console page, you can specify whether users can access specific EJB modules.

EJB module

Specifies the EJB module name.

URI

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Deny all access

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Information

Default:

Value

Cleared

Bind listeners for message-driven beans settings

Use this page to specify bindings for message-driven beans in your application or module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > application_name > Message Driven Bean listener bindings**. This page is the same as the **Bind listeners for message-driven beans** page on the application installation and update wizards.

Each message-driven bean must be bound to a listener port name or to an activation specification Java Naming and Directory Interface (JNDI) name.

Provide a listener port name if your application uses either of the following Java Message Service (JMS) providers:

- WebSphere MQ messaging provider
- Generic messaging provider

Provide an activation specification JNDI name if your application's resources are configured using the default messaging provider or any generic J2C resource adapter that supports inbound messaging.

Not providing valid listener port names or activation specification JNDI names results in the following errors:

- If neither a listener port name or an activation specification JNDI name is specified for a message driven bean, then a validation error is displayed after you click **Finish** on the **Summary** page.
- If multiple message driven beans are linked to the same destination, specify the same destination JNDI name for each message driven bean. If you specify different destination JNDI names, a validation error is displayed and all JNDI specifications after the first one are ignored.

To apply binding changes to multiple mappings:

1. In the list of mappings, select the **Select** check box beside each EJB module that you want mapped to a particular binding.
2. Expand **Apply Multiple Mappings**.
3. Complete one of the following steps:
 - Specify a listener port name.

- Select a target resource JNDI name for an activation specification. Optionally specify the following parameters:

Destination JNDI name

For resource adapters that support JMS, specify `javax.jms.Destinations` so the resource adapter can service messages from the JMS destination. A destination JNDI name set as part of application deployment take precedence over properties set on an activation specification administrative object.

ActivationSpec authentication alias

Specify an authentication alias that is used to access the user name and password that are set on the configured J2C activation specification. Authentication alias properties set as part of application deployment take precedence over properties set on an activation specification administrative object.

4. Click **Apply**.
5. Click **OK** or **Next**.

Module

Specifies the name of the module that contains the enterprise bean.

Bean

Specifies name of an enterprise bean in the application.

URI

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

Messaging Type

Specifies the type of message-driven bean.

Listener Bindings

Specifies a listener port name or an activation specification JNDI name for the message-driven bean. When a message-driven enterprise bean is bound to an activation specification JNDI name you can also specify the destination JNDI name and the authentication alias.

Bindings specify JNDI names for the referenceable and referenced artifacts in an application. An example JNDI name for a listener port to be used by a Store application might be `StoreMdbListener`. The binding definition is stored in IBM bindings files such as `ibm-ejb-jar-bnd.xmi`.

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be `.xmi`.
- For an application or module that uses Java EE 5 or later, the file extension must be `.xml`. If `.xmi` files are included with the application or module, the product ignores the `.xmi` files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the `.xmi` file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the `.xmi` file extensions.

Map data sources for all 2.x CMP beans

Use this page to set the default data source mapping for EJB modules that contain 2.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 2.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console panel, click **Applications > Application Types > Websphere enterprise applications > *application_name* > Map data sources for all 2.x CMP beans** .

This panel displays a table that depicts the EJB modules in your application that contain 2.x CMP beans. Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target of the EJB module only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI Names

Specifies the JNDI name to bind to one or more modules. Select one or more modules, click **Set Multiple JNDI Names**, and select the JNDI name for the resource to which you would like to bind the module.

Set Authorization Type

Specifies the authorization type that you to use for the modules. Select one or more modules, click **Set Authorization Type**, and select the authorization type.

You can choose:

- Per application - indicates that the enterprise bean code performs signon.
- Container - indicates that the application server performs signon to the data source.

Modify Resource Authentication Method

Specifies the resource authentication method for the modules that you have configured with container-managed authorization. Select one or more modules, click **Modify Resource Authentication Method**, and select the authentication method.

You can choose between the following authentication methods:

- **None:**
 1. Determine which data source configurations to designate with no authentication method.
 2. Select the appropriate table rows.
 3. Select **None** from the list of authentication method options that precede the table.
 4. Click **Apply**.
- **Use default method (many-to-one mapping):**
 1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
 2. Select the appropriate table rows.
 3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
 4. Select an authentication data entry or alias from the list.
 5. Click **Apply**.
- **Use Kerberos authentication:** Specifies to use the Kerberos authentication method.
 1. Ensure that you have configured the Kerberos authentication mechanism in the application server.
 2. Select the appropriate table row.
 3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.

4. Select an application login configuration from the list.
5. Click **Apply**.
6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Use trusted connections (one-to-one mapping):**

1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
2. Select the appropriate table row.
3. Ensure that the database to which the modules will connect is configured for trusted connections.
4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.
5. Select an application login configuration from the list.
6. Click **Apply**.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Custom login configuration:**

1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
2. Select the appropriate table row.
3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
4. Select an application login configuration from the list.
5. Click **Apply**.
6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select

Select the check boxes of the rows you want to edit.

EJB Module

Specifies the name of the module that contains the 2.x enterprise beans.

URI

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

JNDI name

Specifies the Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Information

Data type

Value

String

Resource authorization

Specifies the authorization type and the authentication method for securing the data source.

Extended Datasource Properties

When selected, you will be directed to a panel on which you can specify extended properties that the module can use for the DB2 data source.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

Map data sources for all 2.x CMP beans settings

Use this page to map container-managed persistence (CMP) 2.x beans of an application to data sources that are available to the application.

To view this administrative console page, click **Applications > Application Types > Websphere enterprise applications > *application_name* > Map data sources for all 2.x CMP beans**.

Each table row corresponds to a CMP bean within a specific EJB module. A row shows the JNDI name of the data source mapping target of the bean only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI names

Specify the Java Naming and Directory Interface (JNDI) name for multiple EJB modules. Select one or more EJB modules from the table, and select a JNDI name from this list to configure the EJB modules with that JNDI name.

Information	Value
Data type	Drop-down list

Set Authorization Type

Specify the authorization type for securing the data source. Select one or more EJB modules from the table to set the authorization type.

Select either **Container** or **Application** from the displayed list. Container-managed authorization indicates that WebSphere Application Server performs signon to the data source. Application-managed authorization indicates that the enterprise bean code performs signon.

Modify Resource Authentication Method

Specify the authorization type and the authentication method for securing the data source. Select one or more EJB modules from the table to modify the resource authentication method.

You can choose between the following authentication methods:

- **None:**
 1. Determine which data source configurations to designate with no authentication method.
 2. Select the appropriate table rows.
 3. Select **None** from the list of authentication method options that precede the table.
 4. Click **Apply**.
- **Use default method (many-to-one mapping):**
 1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
 2. Select the appropriate table rows.
 3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
 4. Select an authentication data entry or alias from the list.
 5. Click **Apply**.
- **Use Kerberos authentication:** Specifies to use the Kerberos authentication method.
 1. Ensure that you have configured the Kerberos authentication mechanism in the application server.

2. Select the appropriate table row.
3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.
4. Select an application login configuration from the list.
5. Click **Apply**.
6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Use trusted connections (one-to-one mapping):**

1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
2. Select the appropriate table row.
3. Ensure that the database to which the modules will connect is configured for trusted connections.
4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.
5. Select an application login configuration from the list.
6. Click **Apply**.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Custom login configuration:**

1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
2. Select the appropriate table row.
3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
4. Select an application login configuration from the list.
5. Click **Apply**.
6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select

Select the check boxes of the rows that you want to edit.

EJB

The name of an enterprise bean in the application.

EJB Module

The name of the module that contains the enterprise bean.

URI

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

Target resource JNDI name

Specifies the resource to which the CMP bean is bound.

Resource authorization

Specifies the current setting for the resource authorization type.

Modify this setting with **Set authorization type**.

Ensure all unprotected 2.x methods have the correct level of protection

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 2.x methods have the correct level of protection before you map users to roles.

This administrative console page is displayed during the application deployment process. To access the administrative console page, click **Applications > New application > *application_name***. The page is displayed as **Ensure all unprotected 2.x methods have the correct level of protection** in the application deployment steps. On this administrative console page, you can specify whether users can access specific EJB modules.

To use this administrative console page, select the **Uncheck**, **Exclude**, or **Role** option, the check box next to the EJB module, and click **Apply**. If you select **Role** option, select the appropriate role for the EJB module before you click **Apply**.

Uncheck

Select this option if you do not want the application server to verify the access permissions for the EJB module. Everyone can access the EJB module.

Information	Value
Default:	Selected

Exclude

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Information	Value
Default:	Deselected

Role

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the module, then it is added to the modules or Java archive (JAR) files.

Information	Value
Default:	Deselected

EJB module

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Protection type

Specifies the level of protection that is assigned to a particular module name.

After you select the **Uncheck**, **Exclude**, or **Role** option and click **Apply**, the selected protection option is displayed in this column.

Provide options to perform the EJB Deploy settings

Use this page to specify options for the enterprise bean (EJB) deployment tool. The tool generates code needed to run enterprise bean files. You can specify extra class paths, Remote Method Invocation compiler (RMIC) options, database types, and database schema names to be used while running the EJB deployment tool.

This administrative console page is a step in the application installation and update wizards. To view this page, you must select **Deploy enterprise beans** on the **Select installation options** page. Thus, to view this page, click **Applications > New Application > New Enterprise Application > *application_path* > Next > Detailed - Show all installation options and parameters > Next > Deploy enterprise beans > Next > Step: Provide options to perform the EJB Deploy**.

You can specify the EJB deployment tool options on this page when installing or updating an application that contains EJB modules. The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.x modules.


The options that you specify set parameter values for the `ejbdeploy` command. The tool, and thus the `ejbdeploy` command, is run on the enterprise archive (EAR) file during installation after you click **Finish** on the **Summary** page of the wizard.

Class path

Specifies the class path of one or more zipped or Java archive (JAR) files on which the JAR or EAR file being installed depends.

To specify the class paths of multiple entries, the file names must be fully qualified, separated by a path separator that the target server uses, and enclosed in double quotation marks.

```
path\myJar1.jar;path\myJar2.jar;path\myJar3.jar
```

 On the other supported operating systems, the path separator is a colon (:). For example:
`path/myJar1.jar:path/myJar2.jar:path/myJar3.jar`

Class path is the same as the `ejbdeploy` command parameter `-cp class_path`.

Information	Value
Data type	String
Default	null

RMIC

Specifies whether the EJB deployment tool passes RMIC options to the Remote Method Invocation compiler. Refer to RMI Tools documentation for information on the options.

Separate options by a space and enclose them in double quotation marks. For example:

```
"-nowarn -verbose"
```

The **RMIC** setting is the same as the `ejbdeploy` command parameter `-rmic "options"`.

Information	Value
Data type	String
Default	null

Database type

Specifies the name of the database vendor, which is used to determine database column types, mapping information, `Table.sql`, and other information. Select a database type or the empty choice from the drop-down list. The list contains the names of valid database vendors. Selecting the empty choice sets the database type to "" (null).

If you specify a database type, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type. To enable backend IDs for individual EJB modules, select the empty choice to set the database type to null.

Note: The backend IDs SQL92 (1992 SQL Standard) and SQL99 (1999 SQL Standard) are deprecated. Although the SQL92 and SQL99 backend IDs are available in the list on the Provide options to perform the EJB Deploy page, they are deprecated.

Database type is the same as the `ejbdeploy` command parameter `-dbvendor name`.

Information	Value
Data type	String
Default	DB2UDB_V82

Database schema

Specifies the name of the schema that you want to create.

The EJB deployment tool saves database information in the schema document in the JAR or EAR file, which means that the options do not need to be specified again. It also means that when a JAR or EAR is generated, the correct database must be defined at that point because it cannot be changed later.

If the name of the schema contains any spaces, the entire name must be enclosed in double quotes. For example:

```
"my schema"
```

Database schema is the same as the `ejbdeploy` command parameter `-dbschema "name"`.

Information	Value
Data type	String
Default	null

Database access type

Specifies the database access type for a DB2 database that supports Structured Query Language for Java (SQLJ). Use SQLJ to develop data access applications that connect to DB2 databases. SQLJ is a set of programming extensions that support use of the Java programming language to embed statements that provide SQL (Structured Query Language) database requests.

To view this setting, you must select a DB2 backend database that supports SQLJ from the **Database type** drop-down list.

Available database access types include JDBC and SQLJ.

Information	Value
Data type	String
Default	JDBC

SQLJ class path

Specifies the class path of the DB2 SQLJ tool `sqlj.zip` file. The product uses this class path to run the DB2 SQLJ tool during application installation and generate SQLJ profiles (`.ser` files).

To view this setting, you must select a DB2 backend database that supports SQLJ from the **Database type** drop-down list.

Specify the drive and directory where the `sqlj.zip` file resides. For example:

IBM i On all other operating systems, specify `/SQLJ/sqlj.zip`.

When you reinstall an application EAR file, the product deletes any existing SQLJ profiles and creates new profiles.

If you do not specify a class path, the product displays a warning about the missing class path. After you specify a valid class path, you can continue using the wizard for the application installation.

You can customize or add bindings to the generated SQLJ profile after the product installs the application. Use the administrative console SQLJ profiles and pureQuery bind files page accessed by clicking **Applications > Application Types > WebSphere enterprise applications > *application_name* > SQLJ profiles and pureQuery bind files**.

Information	Value
Data type	String
Default	null

JDK compliance level

Specifies the Java developer kit compiler compliance level as *1.4*, *5.0*, *6.0*, or *7.0* when you include application source files for compilation.

The default is to use whatever developer kit version the `ejbdeploy` command is using. For example, if your application is using new functionality defined in Version 7.0 or you are including source files (which is not recommended), then you must specify the Version 7.0 level.

The JDK compliance level that you specify must be the same level as the default Java SDK for the application server to which you are deploying your application. For example, if you select *7.0* for the JDK compliance level on this page, you must ensure that JDK 7.0 is installed and that the default Java SDK for the application server is set to *7.0*. The Java SDKs page of the administrative console lists the software development kits that are installed on the node and enables you to select a default SDK for the node or server. To view the Java SDKs page, click **Servers > Server Types > WebSphere application servers > *server_name* > Java SDKs**.

JDK compliance level is the same as the `ejbdeploy` command parameter `-complianceLevel "1.4" | "5.0" | "6.0" | "7.0"`.

Information	Value
Data type	String
Default	null (empty string)

Shared library reference and mapping settings

Use the Shared library references and Shared library mapping pages to associate defined shared libraries with an application or web module. A shared library is an external Java archive (JAR) file that is used by one or more applications. Using shared libraries enables multiple applications deployed on a server to use a single library, rather than use multiple copies of the same library. After you associate shared libraries

with an application or module, the application or module class loader loads classes represented by the shared libraries and makes those classes available to the application or module.

To view the Shared library references console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Shared library references**. To view the Shared library mapping page, click **Reference shared libraries** on the Shared library references page. These pages are the same as the Map shared libraries and Map shared libraries to an entire application or module pages in the application installation and update wizards.

On the Shared library references page, the first element listed is the application. The other elements are modules in the application.

To associate shared libraries with your application or module:

1. Select an application or module.
2. Click **Reference shared libraries**.
3. On the Shared library mapping page, select one or more shared libraries that the application or modules uses in the **Available** list, click >> to add them to the **Selected** list, and click **OK**.

A defined shared library for a file that your application or module uses must exist to associate your application or module to the library.

If no shared libraries are defined and the application is installed already, on the Shared library mapping page, click **New** and define a shared library.

You can otherwise define a shared library as follows:

1. Click **Environment > Shared libraries**.
2. Specify whether the shared library is visible at the cell, node or server level.
3. Click **New**.
4. On the settings page for the new shared library, specify a name and one or more class paths. If the libraries are platform-specific files such as .dll, .so, or *SRVPGM objects, also specify a native library path. Then, click **Apply**.
5. Save the administrative configuration.

Application

Specifies the name of the application that you are installing or that you selected on the Enterprise applications page.

Module

Specifies the name of the module associated with the shared libraries.

URI

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

Shared libraries

Specifies the name of the shared library files associated with the application or module.

Shared library relationship and mapping settings

Use the Shared library relationship and Shared library relationship mapping pages to specify relationship identifiers and composition unit names for shared libraries that modules in your enterprise application reference. When installing your enterprise application, the product creates a composition unit for each shared library relationship in the business-level application that you specified on the Select installation options page of the application installation wizard.

To view this console page in a wizard, click **Applications > Install new application > New Enterprise Application > *application_path* > Next > Detailed - Show all installation options and parameters > Next > *application_name* > Step: Map shared library relationships.**

After installation, click **Applications > Application Types > WebSphere enterprise applications > Shared library relationships.**

To map library files used in a business-level application to an application or web module, use the Shared library relationship mapping page:

1. Click **Reference shared libraries.**
2. Note the application or module in **Map libraries to the application or module listed.** You are associating library files with that application or module.
3. From the **Available** list, select one or more libraries that the application or module uses.
4. Click >> to add them to the **Selected** list.
5. To remove an association, select one or more libraries in the **Selected** list and click <<.
6. Click **OK.**

Module

Specifies the name of the module associated with the shared libraries.

URI

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

Relationship identifiers

Specifies an identifier for a module shared library relationship. The product assigns an identifier to the composition unit that it creates for the shared library relationship in the business-level application.

Composition unit names

Specifies a composition unit name for the shared library relationship. The product uses this value to name the composition unit that it creates for the shared library relationship in the business-level application that you specified on the Select installation options page of this wizard.

This setting is only in the application installation and update wizards.

Match target

Specifies whether the product maps the composition unit for the shared library relationship to the same deployment target as the business-level application.

Note: If you later change the deployment target of the business-level application or its modules, you must manually update the shared library target to match the target of the application and modules. The targets of shared library composition units are not automatically updated. Not updating the target of the shared library composition unit might cause `java.lang.ClassNotFoundException` errors and prevent the application or its modules from starting. To prevent these error conditions, also ensure that shared libraries upon which other modules or applications depend have a lower starting weight than dependent applications and modules.

JSP and JSF option settings

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

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

The following note applies to the files with an `.xmi` extension in this topic:

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be `.xmi`.
- For an application or module that uses Java EE 5 or later, the file extension must be `.xml`. If `.xmi` files are included with the application or module, the product ignores the `.xmi` files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the `.xmi` file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the `.xmi` file extensions.

Web module

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

URI

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

JSP enable class reloading

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

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

Java Platform, Enterprise Edition 5 (Java EE 5) and later applications IBM extension files are in `.xml` file format. For applications versions earlier than Java EE 5, they are in the `.xmi` file format.

JSP reload interval in seconds

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

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

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

Java EE 5 applications and later IBM extension files are in `.xml` file format. For applications versions earlier than Java EE 5, they are in the `.xmi` file format.

Sun Reference Implementation 1.2

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

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

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

MyFaces 2.0

Select this option to use the MyFaces JSF implementation. This is the default JSF implementation.

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

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

In a mixed-version cell, a Version 7 node uses MyFaces 1.2 if the MyFaces selection is toggled, while a Version 8 and later node uses MyFaces 2.0. For WebSphere Application Server versions before Version 7 (for example, Version 6.1 and earlier), this toggle is ineffective because JSF implementation switching was not supported before Version 7.

Context root for web modules settings

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

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

Web Module

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

URI

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

Context Root

Specifies the context root of the web application (WAR).

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

Initial parameters for servlets settings

Use this page to specify initial parameters that are passed to the init method of web module servlet filters. You can specify initial parameter values for servlets in web modules during or after installation of an application onto a WebSphere Application Server deployment target. The `<param-value>` values specified in `<init-param>` statements in the `web.xml` file of web modules are used by default.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Init parameters for servlets**. This page is the same as the Init parameters for servlets in each web module panel on the application installation and update wizards.

Module

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

URI

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

Servlet

Specifies a unique name for the servlet within the application.

A *servlet* is a Java program that uses the Java Servlet Application Programming Interface (API). You must package servlets in a Web archive (WAR) file or web module for deployment to an application server. Servlets run on a Java-enabled web server and extend the capabilities of a web server, similar to the way applets run on a browser and extend the capabilities of a browser.

Name

Specifies the name of the initial parameter passed to the init method of the web module servlet filter.

The following example servlet filter statement in a `web.xml` file specifies an initial parameter name of `attribute`:

```
<init-param>
  <param-name>attribute</param-name>
  <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
</init-param>
```

Value

Specifies the value assigned to an initial parameter passed to the init method of the web module servlet filter.

The following example servlet filter statement in a `web.xml` file specifies an initial parameter value of `tests.Filter.DoFilter_Filter.SERVLET_MAPPED` for the init parameter `attribute`:

```
<init-param>
  <param-name>attribute</param-name>
  <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
</init-param>
```

Description

Specifies information on the initial parameter.

Environment entries for client modules settings

Use this page to configure the environment entries of application client modules that are deployed as Java archive (JAR) files.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Environment entries for client modules**.

This page is the same as the Map environment entries for client modules page on the application installation and update wizards. To view the Map environment entries for client modules page in a wizard, you must select the **Deploy client modules** option on the Select installation options page.

Client module

Specifies the name of a client module.

URI

Specifies the location of the client module relative to the root of the application.

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the client module.

Type

Specifies a data type for the environment entry defined by the env-entry property in the client module.

Description

Specifies information about the environment entry.

Value

Specifies an editable value for the environment entry. The value is defined by the env-entry property in the client module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for EJB modules settings

Use this page to configure the environment entries of Enterprise JavaBeans (EJB) modules such as entity, session, or message driven beans.

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

Module

Specifies the name of an EJB module.

URI

Specifies the location of the EJB module relative to the root of the application.

Bean

Specifies the name of an enterprise bean that is contained by the module.

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the EJB module.

Type

Specifies a data type for the environment entry defined by the env-entry property in the EJB module.

Description

Specifies information on the environment entry.

Value

Specifies an editable value for the environment entry defined by the env-entry property in the EJB module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for web modules settings

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

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

Module

Specifies the name of a web module.

URI

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

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the web module.

Type

Specifies a data type for the environment entry defined by the env-entry property in the web module.

Description

Specifies information on the environment entry.

Value

Specifies an editable value for the environment entry defined by the env-entry property in the web module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for application settings

Use this page to configure the environment entries of applications that are deployed as enterprise archive (EAR) files.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Environment entries for the application**.

This page is the same as the Map environment entries for application level page on the application installation and update wizards. To view this page, the application must define one or more environment entries.

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the application.

Type

Specifies a data type for the environment entry defined by the env-entry property in the application.

Description

Specifies information about the environment entry.

Value

Specifies an editable value for the environment entry. The value is defined by the env-entry property in the application.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Resource environment references

Use this page to designate how the resource environment references of application modules map to remote resources, which are represented in the product as resource environment entries.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Resource environment references**.

Each row of the table depicts a resource environment reference within a specific module of your application. If you bound any references to resource environment entries during application assembly, you see the JNDI names of those resource environment entries in the applicable rows.

To set the mapping relationships between your resource environment references and resource environment entries:

1. Select a row. Be aware that if you check multiple rows on this page, the resource mapping target that you select in step 2 applies to all of those references.
2. Click **Browse** to select a resource environment entry from the new page that is displayed, the Available Resources page. The Available Resources page shows all resource environment entries that are available mapping targets for your application references.
3. Click **Apply**. The console displays the Resource environment references page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
4. Repeat the previous steps as necessary.
5. Click **OK**. You now return to the general configuration page for your enterprise application.

Table column heading descriptions:

Select

Select the check boxes of the rows that you want to edit.

Module

The name of a module in the application.

EJB

The name of an enterprise bean that is accessed by the module.

URI

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

Reference binding

The name of a resource environment reference that is declared in the deployment descriptor of the application module. The reference corresponds to a resource that is bound as a resource environment entry into the JNDI name space of the application server.

JNDI name

The Java Naming and Directory Interface (JNDI) name of the resource environment entry that is the mapping target of the resource environment reference.

Information

Data type

Value

String

Message destination reference settings

If your application uses message-driven beans, use this page to specify the Java Naming and Directory Interface (JNDI) name of the J2C administered object to bind the message destination reference to the message-driven beans. You must map each message destination reference that is defined in your application to an administered object.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Message destination references**. This page is the same as the **Bind message destination references to administered objects** page on the application installation and update wizards.

If the message destination reference is from an EJB 3.0 or later module, then the JNDI name is optional and the run time provides a container default value.

Attention: If multiple message destination references link to the same message destination, only one JNDI name is collected. When a message destination reference links to the same message destination as a message-driven bean and the destination JNDI name has been collected already, the destination JNDI name for the message destination reference is not collected.

To apply binding changes to multiple mappings:

1. In the list of mappings, select the **Select** check box beside each EJB module that you want mapped to a particular binding.
2. Expand **Apply Multiple Mappings**.
3. Complete one of the following steps:
 - Specify a message destination name.
 - Select a target resource JNDI name for a message destination.
4. Click **Apply**.
5. Click **OK** or **Next**.

Module

Specifies the name of the module that contains the bean.

Bean

Specifies name of a bean in the application.

URI

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

Message destination object

Specifies the message destination object.

Type

Specifies the type of object.

Target Resource JNDI Name

Specifies the Java Naming and Directory Interface (JNDI) name of the bean.

This is a data entry field. To change the JNDI name bound to this bean, type the new name in this field.

Select current backend ID settings

Use this page to select a backend identifier for container-managed persistence (CMP) beans that contain mappings for multiple backend databases.

This administrative console page is a step in the application installation and update wizards. To view this administrative console page, click **Applications > New Application > New Enterprise Application > application_path > Next > Detailed - Show all installation options and parameters > Next > Next** or **Continue > Step: Select current backend ID**. This page is displayed in the wizards if **Database type** is blank on the Provide options to perform the EJB Deploy page.

A backend can represent different database vendors, or simply alternative mappings and table qualifiers. If a Java archive (JAR) file for an enterprise bean defines CMP beans that contain mappings for multiple backend databases, you must select a current backend ID to be used when the module is installed on a deployment target. The backend ID determines the persister classes that get loaded at deployment.

Module

Specifies the name of the module that contains the bean.

URI

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

Current backend ID

Specifies the current backend ID to be used when the module is installed on a deployment target.

Provide JNDI names for JCA objects settings

Use this page to configure Java Naming and Directory Interface (JNDI) name values for J2C objects (J2CConnectionFactory, J2CActivationSpec, and J2CAdminObject) in your application or modules. If your application contains an embedded resource archive (RAR) file, specify the name and JNDI name of each JCA connection factory, administered object, and activation specification.

This administrative console page is a step in the application installation and update wizards. To view this administrative console page, click **Applications > New Application > New Enterprise Application > application_path > Next > Detailed - Show all installation options and parameters > Next > Next** or **Continue > Step: Provide JNDI names for JCA objects**.

Connector module

Specifies the name of a connector module of the RAR file.

URI

Specifies the location of the module that is relative to the root of the RAR file.

Object identifier

Specifies the name of the J2C object. The object can be a JCA connection factory, administered object, or activation specification.

Bindings

Specifies the name and Java Naming and Directory Interface (JNDI) name of the J2C object.

These are data entry fields. To change the name or JNDI name bound to this object, type the new names in the fields.

Correct use of the system identity

Use this page to manage the system identity properties for the Enterprise JavaBeans (EJB) method in your application.

This administrative console page is displayed during the application deployment process. To access the administrative console, click **Application > New application > New Enterprise Application**. This is displayed as *Correct use of System Identity* in the application deployment steps.

To use this page, complete the following steps:

1. Select an application that supports security and click **Next**.
2. Select **Detailed - Show all installation options and parameters** and click **Next**.
3. Select the **Correct use of system identity** step.

Bean

A component that implements a business task or business entity and resides in an EJB container. Entity beans, session beans, and message-driven beans are all enterprise beans.

Module

In Java EE programming, a software unit that consists of one or more components of the same container type and one deployment descriptor of that type. Examples include EJB, Web, and application client modules.

URI

A Uniform Resource Identifier (URI) is a unique address that is used to identify content on the Web, such as a page of text, a video or sound clip, a still or animated image, or a program.

Method signature

The combination of a name of a method along with the number and types of the parameters and their order.

Role

Specifies the RunAs role that is used for this EJB method.

Username

Specifies the user name that is assigned to the RunAs role for this EJB method.

The user name is used in conjunction with the RunAs role that you select for the Role.

Requirements for setting data access isolation levels

This article discusses the criteria and effects of setting isolation levels for data access components that comprise Enterprise JavaBeans (EJB) 2.x and later modules.

In an EJB 1.1 module, you can set the isolation level at the method level or bean level. This capability also applies to container-managed persistence (CMP) 1.1 beans that you assemble into *EJB 2.x modules*. WebSphere Application Server permits the deployment descriptor of a CMP bean to declare the version level of 1.1, regardless of the overall module version.

However, the ability to set isolation level at the method or bean level does **not** apply to other enterprise beans within an EJB 2.x module, including *CMP 2.x beans*. WebSphere Application Server Version 5.0 removed this capability from EJB 2.0 modules to deliver an architecture that ultimately provides more efficient connection use.

Consequently, later versions of the product enforce the following restrictions on declaring isolation level for CMP 2.x beans—as well as session beans, message-driven beans, and bean managed persistence (BMP) beans that you assemble into EJB 2.x modules:

- You cannot specify isolation level on the EJB method level or bean level.
- If you configure a JDBC application, a bean-managed persistence (BMP) bean, or a servlet to participate in global transactions, any connection that is shared cannot accept a user-specified isolation level. WebSphere Application Server can only set a user-specified isolation level on a connection that is not shared within a global transaction. *Generally, you want to refrain from specifying isolation levels on shareable connections.*

The configuration for the isolation level is determined by the type of bean that is used by the component:

Isolation level on connections used by 2.x CMP beans

In a EJB 2.x module, when a CMP 2.x bean uses a new data source to access a backend database, the isolation level is determined by the WebSphere Application Server run time, based on the type of access intent assigned to the bean or the calling method. Other non-CMP connection users can access this same data source and also use the access intent and application profile support to manage their concurrency control.

Connections used by other 2.x enterprise beans and other non-CMP components

For all other JDBC connection instances (connections other than those used by CMP beans), you can specify an isolation level on the data source resource reference. For shareable connections that run in global transactions, this method is the only way to set the *isolationLevel* for connections. Trying to directly set the isolation level through the *setTransactionIsolation()* method on a shareable connection that runs in a global transaction is not allowed. To use a different isolation level on connections, you must provide a different resource reference. Set these defaults through your assembly tool.

Each resource reference associates with one isolation level. When your application uses this resource reference Java Naming and Directory Interface (JNDI) name to look up a data source, every connection returned from this data source using this resource reference has the same isolation level.

Components needing to use shareable connections with multiple isolation levels can create multiple resource references, giving them different JNDI names, and have their code look up the appropriate data source for the isolation level they need. In this way, you use separate connections with the different isolation levels enabled on them.

It is possible to map these multiple resource references to the same configured data source. The connections still come from the same underlying pool, however; the connection manager does not allow sharing of connections requested by resource references with different isolation levels.

Consider the following scenario:

- A data source is bound to two resource references: *jdbc/RRResRef* and *jdbc/RResRef*.
- *RRResRef* has the *RepeatableRead* isolation level defined. *RResRef* has the *ReadCommitted* isolation level defined.

If your application wants to update the tables or a BMP bean updates some attributes, it can use the *jdbc/RRResRef* JNDI name to look up the data source instance. All connections returned from the data source instance have a *RepeatableRead* isolation level. If the application wants to perform a query for read only, then it is better to use the *jdbc/RResRef* JNDI name to look up the data source.

The product does not require you to set the isolation level on a data source resource reference for a non-CMP application module. If you do not specify isolation level on the resource reference, or if you specify *TRANSACTION_NONE*, the WebSphere Application Server run time uses a default isolation level for the data source. Application Server uses a default setting based on the JDBC driver.

For most drivers, WebSphere Application Server uses an isolation level default of *TRANSACTION_REPEATABLE_READ*. For Oracle drivers, however, Application Server uses an isolation level of *TRANSACTION_READ_COMMITTED*. Use the following table for quick reference:

Database:	Default isolation level:
DB2	RR
Oracle	RC
Sybase	RR
Informix	RR
Apache Derby	RR

Database:	Default isolation level:
SQL Server	RR

Note: These same default isolation levels are used in cases of direct JNDI lookups of a data source.

- RR = JDBC Repeatable read (TRANSACTION_REPEATABLE_READ)
- RC = JDBC Read committed (TRANSACTION_READ_COMMITTED)

To customize the default isolation level, you can use the `webSphereDefaultIsolationLevel` custom property for the data source. In most cases you should define the isolation level in the deployment descriptor when you package the EAR file, but in certain situations you might need to customize the default isolation level. This property will have no effect if any of the previous options are used, and this custom property is provided for those situations in which there is no other means of setting the isolation level.

Use the following values for `webSphereDefaultIsolationLevel` custom property:

Possible values	JDBC isolation level	DB2 isolation level
8	TRANSACTION_SERIALIZABLE	Repeatable Read (RR)
4 (default)	TRANSACTION_REPEATABLE_READ	Read Stability (RS)
2	TRANSACTION_READ_COMMITTED	Cursor Stability (CS)
1	TRANSACTION_READ_UNCOMMITTED	Uncommitted Read (UR)

To define this custom property for a data source:

1. Click **Resources > JDBC provider > JDBC_provider**.
2. Click **Data sources** in the Additional Properties section.
3. Click the name of the data source.
4. Click **Custom properties**.
5. Create the `webSphereDefaultIsolationLevel` custom property.
 - a. Click **New**.
 - b. Enter `webSphereDefaultIsolationLevel` for the name field.
 - c. Enter one of the possible values in the value field.

Application Server sets the isolation level by prioritizing the available settings. Application Server will set the isolation level based on the values for the following, in this order:

1. Resource reference isolation level
2. Isolation level that is specified by the access intent policy
3. Custom property that configures an isolation level
4. Application Server's default setting.

Metadata for module settings

Use this page to instruct a Java Platform, Enterprise Edition (Java EE) enterprise bean (EJB) deployment descriptor, web module deployment descriptor, or JCA resource adapter archive (RAR) module to ignore annotations that specify deployment information.

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

If your application contains Java EE 5 or later modules, you can select to lock the deployment descriptor of one or more of the modules on the Metadata for modules page. If you select a **metadata-complete**

attribute check box (set the `metadata-complete` attribute to `true`) and lock deployment descriptors, the product writes the complete module deployment descriptor, including deployment information from annotations, to XML format.

Annotations are a standard mechanism of adding metadata to Java classes. You can use metadata to simplify development and deployment of Java EE 5 or later artifacts. Prior to the introduction of Java language annotations, deployment descriptors were the standard mechanism used by Java EE components. These deployment descriptors were mapped to XML format, which facilitated their persistence. If you select to lock deployment descriptors, the product merges Java EE annotation-based metadata with the XML-based existing deployment descriptor metadata and persists the result.

When applications contain a large number of Java classes, the deployment processing time for the annotations can increase. To minimize the performance impact, you can use one of the following methods:

- Determine whether the module needs to use Java EE 5 or 6. If the module does not need to use Java EE 5 or 6, the annotations within the Java classes are not scanned.
- Use the “`metadata-complete` attribute” in the module descriptor if the module uses Java EE 5 or later and it does not contain any annotations. This attribute disables the annotations processing for the module, but Java EE 5 or later modules might still be placed in the descriptor file. If you are migrating your application, but you are not adding annotations, consider using this attribute value.
- Restructure the application to place the utility Java archive (JAR) files into shared libraries if those JAR files do not have annotation information. Consider this method if you cannot set the “`metadata-complete` attribute.”
- Move the JAR files in the `WEB-INF/lib` directory to the root directory of the enterprise archive (EAR) file. Nested archives, such as a JAR file that is within a web application archive (WAR) that is within an EAR file, are very cumbersome to search through because of the multiple levels of compression.

Module

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

Information	Value
Data type	String

URI

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

Information	Value
Data type	String

metadata-complete attribute

Specifies whether to write the complete module deployment descriptor, including deployment information from annotations, to extensible markup language (XML) format.

By default, a **metadata-complete attribute** check box is not selected and the product does not write out annotation data to a module deployment descriptor.

If your modules do not have a `metadata-complete` attribute or the `metadata-complete` attribute is set to `false`, you can select a check box and instruct the product to write out annotation data to a module deployment descriptor.

Note: If your Java EE 5 or later application uses annotations and a shared library, do not select **metadata-complete attribute**. When your application uses annotations and a shared library, setting the `metadata-complete` attribute to `true` causes the product to incorrectly represent an `@EJB` annotation in the deployment descriptor as `<ejb-ref>` rather than `<ejb-local-ref>`. For web modules, setting the `metadata-complete` attribute to `true` might cause `InjectionException` errors. If

you must select **metadata-complete attribute** (set the `metadata-complete` attribute to `true`), avoid errors by not using a shared library, by placing the shared library in either the `classes` or `lib` directory of the application server, or by fully specifying the metadata in the deployment descriptors.

After you select a check box, you cannot deselect (clear) the check box and the module is no longer shown in the list of modules on this page. If you select all the check boxes, the link to this page is no longer shown on the enterprise application settings page.

Information	Value
Data type	Boolean
Default	false (deselected)

Provide options to perform the web services deployment settings

Use this page to specify options for web services deployment.

This administrative console page is a step in the application installation and update wizards.

To view this page, you must select **Deploy web services** on the **Select installation options** page.

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

1. Click **Applications > New application > *application_path***.
2. Select the option to **Show all installation options and parameters**.
3. Click **Next** to get to the **Step: Select installation options** page.
4. Select **Deploy web service**.
5. Click **Next** to get to the **Step: Provide options to perform the web services deployment** page.

You can specify the web services deployment options on this page only when installing or updating an application that uses web services.

The `wsdeploy` command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the `wsdeploy` command. If your web services application contains only JAX-WS endpoints, you do not need to run the `wsdeploy` command, as this command is used to process only JAX-RPC endpoints.

The options that you specify set parameter values for the `wsdeploy` command. The `wsdeploy` command adds product-specific deployment classes to a web services-compatible enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs
- Serializers and deserializers
- Implementations of service interfaces

The `wsdeploy` command is run during installation after you click **Finish** on the **Summary** page of the wizard.

Deploy web services option - Classpath

Specifies entries to add to the CLASSPATH when the generated classes are compiled.

To specify the class paths of multiple entries, you need to separate the entries with a semicolon on Windows platforms and on Linux, Unix, and z/OS platforms, you need to use a colon to separate the entries. This is the same separator that is used with the CLASSPATH environment variable.

This option is the same as the `wsdeploy` command parameter `-cp class_path`.

Information	Value
Data type	String
Default	null

Deploy web services option - Extension Directories

Specifies a directory that contains zipped or Java archive (JAR) files. All zipped and JAR files in this directory are added to the CLASSPATH used to compile the generated files.

This option is the same as the `wsdeploy` command parameter `-jardir directory`.

Information	Value
Data type	String
Default	null

Display module build ID settings

Use this page to view the build identifier of a module in a Java Platform, Enterprise Edition (Java EE) enterprise archive (EAR file). The build identifier for a module is shown if the MANIFEST.MF file of a module or application specifies a build identifier.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Display module build IDs**. This page is the same as the Display module build IDs page on the application installation and update wizards.

Module

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

Information	Value
Data type	String

URI

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

Information	Value
Data type	String

Build ID

Specifies the build identifier for a module if the MANIFEST.MF file specifies a build identifier.

You cannot modify the build ID on this page because this field is read-only.

Information	Value
Data type	String

Task overview: Assembling applications using remote request dispatcher

Remote request dispatcher (RRD) is a pluggable extension to the web container which allows application frameworks, servlets and JavaServer Pages (JSP) to include content from outside the currently executing resource's Java Virtual Machine (JVM) as part of the response sent to the client.

Before you begin

You must have WebSphere Application Server, Network Deployment installed to use remote request dispatcher function. You should also familiarize yourself the limitations of remote request dispatcher. See article, Remote request dispatcher considerations for details.

Procedure

1. Install enterprise application files with the console.
2. Configure the sending of include requests between the application and remote resources.
 - Configure web applications to dispatch remote includes.
 - Configure web applications to service remote includes.
3. Optional: Modify your application to locate resources located in two different contexts using the servlet programming model.

The Servlet Programming Model for including resources remotely does not require you to use any non-Java Platform, Enterprise Edition (Java EE) Servlet Application Programming Interfaces (APIs). The remote request dispatcher (RRD) component follows the same rules to obtain a ServletContext and a remote resource. By using JavaServer Pages standard tag library (JSTL), your application is further removed from obtaining a ServletContext object or RequestDispatcher that is required in the framework example in the following step because the JSTL custom tag does this implicitly. Study the following example of a sample JavaServer Pages application to learn how to locate resources that are in two different contexts, investments and banking.

```
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
isELIgnored="false"
%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
</HEAD>
<BODY>

<%--
```

Programming example using JavaServer Pages and JavaServer Pages Standard Tag Library (JSTL).
JSTL provides a custom tag to import contents (in servlet and JSP terms include) in the scope of the same request from outside of the current web module context by specifying a context parameter.

JSTL restriction: The web module that is imported must run inside of the same JVM as the calling resource if imported URL is not fully qualified.

RRD extends this functionality by permitting the web module to be located within the scope of the current WebSphere Application Server core group versus the scope of the JVM.
--%>

```
<hr size="5"/>
<%-- Include resource investmentSummary.jsp located in the
web application with context root of /investments. --%>

<c:import url="investmentSummary.jsp" context="/investments"/>

<hr size="5"/>
<%-- Include resource accountSummary.jsp located in the
web application with context root of /banking. --%>

<c:import url="accountSummary.jsp" context="/banking"/>

<hr size="5"/>

</BODY>
</HTML>
```

4. Optional: Modify your application to locate resources located in two different contexts using the framework programming model.

The Framework Programming Model for including resources remotely does not require you to use any non-Java Platform, Enterprise Edition (Java EE) Servlet Application Programming Interfaces (APIs). When a request is initiated for a ServletContext name that is not presently running inside of the current

web container, the remote request dispatcher (RRD) component returns a ServletContext object that can locate a resource that exists anywhere inside a WebSphere Application Server WebSphere Application Server, Network Deployment environment provided that the resource exists and RRD is enabled for that ServletContext object. Study the following sample framework snippet that demonstrates how to locate resources located in two different contexts, investments and banking.

```
/*
Programming example using a generic framework.
Servlet Specification provides an API to obtain
a servlet context in the scope of the same request
different from the current web module context by
specifying a context parameter.

Servlet Specification restriction: The web module that obtain
must run inside of the same JVM as the calling resource.

RRD extends this functionality by permitting the web module to be located
within the scope of the current WebSphere Application Server core group
versus the scope of the JVM.
*/

protected void frameworkCall (ServletContext context, HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

    PrintWriter writer = response.getWriter();

    writer.write("<HTML>");
    writer.write("<HEAD>");
    writer.write("</HEAD>");
    writer.write("<BODY>");
    writer.write("<hr size=\"5\">");

    //Include resource investmentSummary.jsp located in web application
    //with context root of /investments.
    RequestDispatcher rd = getRequestDispatcher ( context, "/investments", "/investmentSummary.jsp");
    rd.include(request, response);

    writer.write("<hr size=\"5\">");

    //Include resource accountSummary.jsp located in web application
    //with context root of /banking.
    rd = getRequestDispatcher ( context, "/banking", "/accountSummary.jsp");
    rd.include(request, response);

    writer.write("</BODY>");
    writer.write("</HTML>");
}

private RequestDispatcher getRequestDispatcher (ServletContext context, String contextName, String resource) {
    return context.getContext(contextName).getRequestDispatcher(resource);
}
}
```

Results

After enabling at least one enterprise application to dispatch remote includes and at least one enterprise application to service remote includes, RRD is now enabled.

What to do next

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

Installing enterprise modules with JSR-88

You can install Java Platform, Enterprise Edition (Java EE) modules on an application server provided by a WebSphere Application Server product using the Java EE Application Deployment API specification (JSR-88).

Before you begin

Note: Application installation using the Java EE Application Deployment API specification (JSR-88) was deprecated in WebSphere Application Server Version 8.0. Use another option to deploy applications

to a server. The closest option to using the Java EE Deployment API is using Java Management Extensions (JMX) MBean programming. For information on deployment options, see “Ways to install enterprise applications or modules.”

JSR-88 defines standard application programming interfaces (APIs) to enable deployment of Java EE applications and stand-alone modules to Java EE product platforms. The Java EE Application Deployment specification Version 1.1 is available at <http://java.sun.com/j2ee/tools/deployment/reference/docs/index.html> as part of the Java 2 Platform, Enterprise Edition (J2EE) 1.4 Application Server Developer Release.

Read about JSR-88 and APIs used to manage applications at <http://java.sun.com/j2ee/tools/deployment/>.

About this task

JSR-88 defines a contract between a tool provider and a platform that enables tools from multiple vendors to configure, deploy and manage applications on any Java EE product platform. The tool provider typically supplies software tools and an integrated development environment (IDE) for developing and assembly of Java EE application modules. The Java EE platform provides application management functions that deploy, undeploy, start, stop, and otherwise manage Java EE applications.

WebSphere Application Server is a Java EE specification-compliant platform that implements the JSR-88 APIs. Complete the following steps to deploy (install) Java EE modules on an application server provided by the WebSphere Application Server platform.

Procedure

1. Code a Java program that can access the JSR-88 DeploymentManager class for the product.
 - a. Write code that finds the JAR manifest attribute J2EE-DeploymentFactory-Implementation-Class. Under JSR-88, your code finds the DeploymentFactory using the JAR manifest attribute J2EE-DeploymentFactory-Implementation-Class. The following product application management JAR files contain this attribute and provide support.

Table 27. JAR files that contain the manifest attribute. Enable your code to find the DeploymentFactory using the JAR manifest attribute.

Environment	JAR file containing the manifest attribute
Application server	<i>app_server_root</i> /plugins/com.ibm.ws.admin.services.jar
Application client	<i>app_client_root</i> /plugins/com.ibm.ws.j2ee.client.jar
Thin application client	<i>app_client_root</i> /runtimes/com.ibm.ws.admin.client_8.0.0.jar

After your code finds the DeploymentFactory, the deployment tool can create an instance of the WebSphere DeploymentFactory and register the instance with its DeploymentFactoryManager.

Example code for the application server environment follows. The example code requires that you use the development kit shipped with the product or use the pluggable client for deployment of stand-alone modules. See *WebSphere Application Server detailed system requirements* at <http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921> for information on supported development kits.

```
import javax.enterprise.deploy.shared.factories.DeploymentFactoryManager;
import javax.enterprise.deploy.spi.DeploymentManager;
import javax.enterprise.deploy.spi.factories.DeploymentFactory;
import java.util.jar.JarFile;
import java.util.jar.Manifest;

// Get the DeploymentFactory implementation class from the MANIFEST.MF file.
File jsr88Jar = new File(wasHome + "/plugins/com.ibm.ws.admin.services.jar");
JarFile jarFile = new JarFile(jsr88Jar);
Manifest manifest = jarFile.getManifest();
Attributes attributes = manifest.getMainAttributes();
```

```

String key = "J2EE-DeploymentFactory-Implementation-Class";
String className = attributes.getValue(key);
// Get an instance of the DeploymentFactoryManager
DeploymentFactoryManager dfm = DeploymentFactoryManager.getInstance();

// Create an instance of the WebSphere Application Server DeploymentFactory.
Class deploymentFactory = Class.forName(className);
DeploymentFactory deploymentFactoryInstance =
    (DeploymentFactory) deploymentFactory.newInstance();

// Register the DeploymentFactory instance with the DeploymentFactoryManager.
dfm.registerDeploymentFactory(deploymentFactoryInstance);

// Provide WebSphere Application Server URI, user ID, and password.
// For more information, see the step that follows.
wsDM = dfm.getDeploymentManager(
    "deployer:WebSphere:myserver:8880", null, null);

```

- b. Write code that accesses the DeploymentManager instance for the product.

The product URI for deployment has the following format:

```
"deployer:WebSphere:host:port"
```

The example in the previous step, "deployer:WebSphere:myserver:8880", tries to connect to host myserver at port 8880 using the SOAP connector, which is the default.

You can specify an Internet Protocol Version 6 (IPv6) address for the *host* element in the URI for deployment. Enclose the IPv6 address in square brackets ([]); for example:

```
"deployer:WebSphere:[IPv6_address]:port"
```

Also, you can add an optional parameter, *connectorType*, to the URI for deployment. For example, to use the RMI connector to access myserver, code the URI as follows:

```
"deployer:WebSphere:myserver:2809?connectorType=RMI"
```

2. Optional: Code a Java program that can customize or deploy Java EE applications or modules using the JSR-88 support provided by the product.
3. Start the deployed Java EE applications or stand-alone Java EE modules using the JSR-88 API used to start applications or modules.

What to do next

Test the deployed applications or modules. For example, point a web browser at the URL for a deployed application and examine the performance of the application. If necessary, update the application.

Customizing modules using DConfigBeans

You can configure Java Platform, Enterprise Edition (Java EE) applications or stand-alone modules during deployment using the DConfigBean class in the Java EE Application Deployment API specification (JSR-88).

Before you begin

Note: Application installation using the Java EE Application Deployment API specification (JSR-88) was deprecated in WebSphere Application Server Version 8.0. Use another option to deploy applications to a server. The closest option to using the Java EE Deployment API is using Java Management Extensions (JMX) MBean programming. For information on deployment options, see “Ways to install enterprise applications or modules.”

This topic assumes that you are deploying (installing) Java EE modules on an application server provided by the product using the WebSphere Application Server support for JSR-88.

Read about the JSR-88 specification and using the DConfigBean class at <http://java.sun.com/j2ee/tools/deployment/>.

About this task

The DConfigBean class in JSR-88 provides JavaBeans-based support for platform-specific configuration of J2EE applications and modules during deployment. Your code can inspect DConfigBean instances to get platform-specific configuration attributes. The DConfigBean instances provided by WebSphere Application Server contain a single attribute which has an array of java.util.Map objects. The map entries contain configuration attributes, for which your code can get and set values.

Procedure

1. Write code that installs Java EE modules on an application server using JSR-88.
2. Write code that accesses DConfigBeans generated by the product during JSR-88 deployment. You (or a deployer) can then customize the accessed DConfigBeans instances.

The following pseudocode shows how a Java EE tool provider can get DConfigBean instance attributes generated by the product during JSR-88 deployment and set values for the attributes.

```
import javax.enterprise.deploy.model.*;
import javax.enterprise.deploy.spi.*;
{
DeploymentConfiguration dConfig = ___; // Get from DeploymentManager
DDBeanRoot ddRoot = ___;           // Provided by J2EE tool

// Obtain root bean.
DConfigBeanRoot dcRoot = dConfig.getDConfigBeanRoot(dr);

// Configure DConfigBean.
configureDCBean (dcRoot);
}

// Get children from DConfigBeanRoot and configure each child.
method configureDCBean (DConfigBean dcBean)
{
// Get DConfigBean attributes for a given archive.
BeanInfo bInfo = Introspector.getBeanInfo(dcBean.getClass());
IndexedPropertyDescriptor ipDesc =
    (IndexedPropertyDescriptor)bInfo.getPropertyDescriptors()[0];

// Get the 0th map.
int index = 0;
Map map = (Map)
    ipDesc.getIndexedReadMethod().invoke
        (dcBean, new Object[]{new Integer(index)});

while (map != null)
{
// Iterate over the map and set values for attributes.

// Set the map back into the DCBean.
ipDesc.getIndexedWriteMethod().invoke
    (dcBean, new Object[]{new Integer(index), map});

// Get the next entry in the indexed property
map = (Map)
    ipDesc.getIndexedReadMethod().invoke
        (dcBean, new Object[]{new Integer(++index)});
}
}
```

Chapter 13. Deploying and administering business-level applications

Deploying a business-level application consists of creating the business-level application on a Version 7.0 or later server.

Before you begin

A business-level application is an administration model that provides the entire definition of an application as it makes sense to the business. It is a WebSphere configuration artifact, similar to a server or cluster, that is stored in the product configuration repository. A business-level application can contain artifacts such as Java Platform, Enterprise Edition (Java EE) applications or modules, shared libraries, data files, and other business-level applications. You might use a business-level application to group related artifacts or to add capability to an existing application. For example, suppose you want to add capability provided in a Java archive (JAR) to a Java EE application already deployed on a product server. You can add that capability by creating a new business-level application and adding the JAR file and the deployed Java EE application to the business-level application. In some cases, you do not even need to change the deployed Java EE application configuration to add the capability.

Before creating a business-level application, you must develop the artifacts to go in the application and configure the target server or cluster. Before choosing a deployment target for the application, ensure that the target version is 7.0 or later.

About this task

When creating a business-level application, you can configure the application enough to enable it to run on the server. Later, you can configure the application and its contents further, start or stop the application, and otherwise manage its activity.

The topics in this section describe how to deploy and administer a business-level application or its contents using the administrative console. You can also use programming or wsadmin scripting.

Procedure

- Import assets to a repository.
- View, delete, update, or export assets.
- Create a business-level application.
- Start the application.
- Stop the application.
- Update the application and its configuration units.
- Delete the application.

What to do next

After making changes to administrative configurations of your applications in the administrative console, ensure that you save the changes.

Business-level applications

A business-level application is an administration model that provides the entire definition of an application as it makes sense to the business. A business-level application is a WebSphere configuration artifact, similar to a server or cluster, that is stored in the product configuration repository.

- Business-level application characteristics

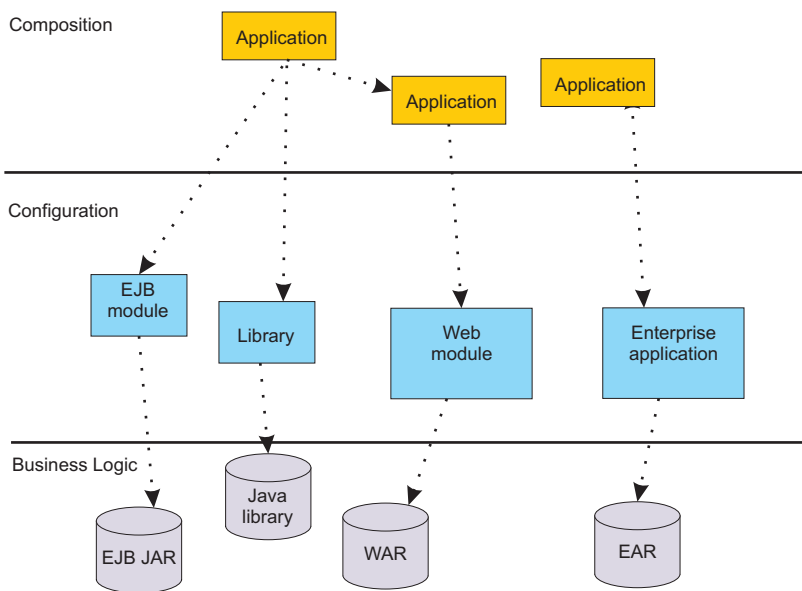
- Comparisons to Java EE applications

Business-level application characteristics

A business-level application has the following characteristics:

- A business-level application is an administration model of the definition of an enterprise-level application that consists of WebSphere and non-WebSphere artifacts. The business-level application might not explicitly manage the lifecycle of every artifact. It is a model for defining an application.
- A business-level application does not represent or contain application binary files. It is a configuration that lists one or more composition units, which represent the application binary files. A business-level application uses the binary files to run the application business logic. Administration of binary files is separate from administration of the application definition.
- A business-level application supports recursive composition by reference that facilitates hierarchical assembly of business-level applications and individual deployed artifacts within or outside a WebSphere product. The composition at its lowest level consists of configured instances of application binary files that run in a specific runtime environment such as an application server. Installable packages or archives, such as Java archives (JAR) or enterprise archive (EAR) files, typically deliver the business logic that these configured instances represent to corresponding runtime platforms.

The following diagram shows the composition model for business-level applications:



A business-level application does not introduce new programming, runtime, or packaging models:

- You do not need to change your application business logic. The business-level application function does not introduce new application programming interfaces (APIs).
- You do not need to change your application runtime settings. The product supports all of the runtime characteristics, such as security, class loading and isolation, required by individual programming models to which business components are written.
- You do not need to change your application packaging. There is no specific unique packaging model that provides a business-level application definition.

Typically, you first create an empty business-level application and then add composition units to it. The business-level application name must be unique within a cell. The business level application itself has minimal configuration data associated with it, solely the list of composition units, but individual composition units might save application-specific configuration data.

A business-level application is defined in the product configuration repository under `profile_root/config/cells/cell_name/b1a/business_level_application_name/bver/BASE/b1a.xml`.

Comparisons to Java EE applications

Business-level applications can consist of or aggregate Java Platform, Enterprise Edition (Java EE) applications and modules with non-Java EE artifacts. The contents of Java EE applications integrate with business-level application concepts for deployment and management of applications. Existing Java EE application management APIs continue to work after you add Java EE application or modules to a business-level application. The business-level application management API accepts Java EE contents and configurations and delegates to existing Java EE management APIs. Control operations such as starting and stopping a Java EE composition unit are delegated to `ApplicationManager` MBean on application servers that start and stop Java EE applications.

Table 28. Java EE concepts compared to business-level application concepts. Business-level application concepts include assets, composition units, and deployable units.

Java EE concept	Business-level application concept	Description
EAR or stand-alone module for deployment	Asset	Java EE application contents are assets.
Java EE application created at the end of application install	Composition unit	A Java EE application is in an enterprise archive (EAR) file. The product saves the EAR file in the product repository as a composition unit.
Java EE modules within the EAR file	Deployable units in the asset	Each module in the EAR file is a deployable unit that you can install on independent deployment targets. The EAR file is still managed as a single asset in its entirety.
Java EE application installation using the administrative console, programming, or <code>wsadmin</code> commands	Multiple business-level application management commands During Java EE application deployment, you can specify the name of the business-level application to include the Java EE application. If the business-level application name is not set, the product creates a default business-level application with the same name as the Java EE application name. The product adds a composition unit with the same name as the Java EE application name under the business-level application. You can deploy multiple Java EE applications under a single business-level application.	<p>You can make a Java EE application a business-level application and add it to another business-level application:</p> <ol style="list-style-type: none"> 1. Install the Java EE application (EAR file) using the enterprise application installation console wizard, programming, or <code>wsadmin</code>. Keep the default selection to create a business-level application that has the same name as the Java EE application. 2. Create an empty business-level application. 3. Add the EAR file business-level application to the empty business-level application. The EAR file business-level application is a composition unit of the containing business-level application. <p>Or, you can make a Java EE application an asset and add it to another business-level application:</p> <ol style="list-style-type: none"> 1. Import an EAR file as an asset. It has an asset type aspect of Java EE ear. 2. Create an empty business-level application. 3. Add the Java EE application asset to the business-level application. The EAR file asset is a composition unit of the containing business-level application. 4. Collect targets for each deployable unit (Java EE module).
Uninstall Java EE application	Multiple business-level application management commands	<p>You delete the Java EE application composition unit from the business-level application:</p> <ol style="list-style-type: none"> 1. Remove the composition unit for the Java EE application from the business-level application. 2. If the EAR file is an asset, delete the asset.

Table 28. Java EE concepts compared to business-level application concepts (continued). Business-level application concepts include assets, composition units, and deployable units.

Java EE concept	Business-level application concept	Description
Start the Java EE application.	Start the composition unit.	Starting a business-level application starts any Java EE application in it.
Stop the Java EE application.	Stop the composition unit.	Stopping a business-level application stops any Java EE application in it.

Assets

An asset represents one or more application binary files that are stored in an asset repository. Typical assets include application business logic such as Java Platform, Enterprise Edition (Java EE) archives, library files, and other resource files.

An asset repository stores the binary files for the asset. The product configuration repository provides a default asset repository.

Assets in the configuration repository are managed by the product management domain. The configuration repository stores asset binary files in `app_server_root/config/cells/cell_name/assets/asset_name/aver/BASE/bin/`.

An asset name must be unique within a cell, the product administrative domain.

The product creates an `asset.xml` file when an asset is registered with the product configuration. The file contains information about the asset such as its name, destination location, and dependencies on other assets.

You must register files as assets before you can add them to one or more business-level applications. At the time of asset registration, you can import the physical application files into the product configuration repository or you can specify an external location where the asset resides.

Composition units

A composition unit represents a configured asset in a business-level application. A composition unit enables the asset contents to interact with other assets in the application. It also enables the product run time to load and run asset contents.

The product supports three types of composition units:

Asset composition units

Composition units created from assets by configuring each deployable unit of the asset to run on deployment targets.

Shared library composition units

Composition units created from JAR-based assets by ignoring all the deployable objects from the asset and treating the asset JAR file as a library of classes.

Business-level application composition units

Composition units created from business-level applications that are added to existing business-level applications.

A composition unit contains the following information:

- Configuration information that binds contents of an asset with a specific hosting run time and adds the configuration necessary for the run time to load and run the asset
- References to external services, components, or other resources that the asset uses

- Customized configurations for service definitions, references and other relevant configuration data
- A list of deployment targets or runtime environments along with the runtime environment-specific configuration where the composition unit runs.

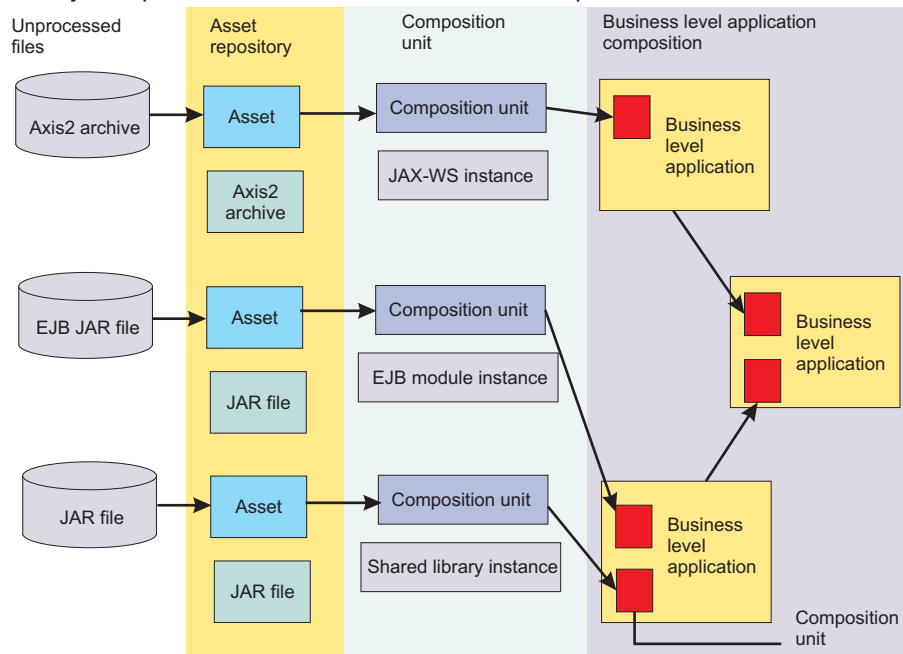
For example, a composition unit for an enterprise bean (EJB) Java archive (JAR) asset is an EJB module instance that contains necessary EJB binding information, such as EJB Java Naming and Directory Interface (JNDI) names and `ejb-ref` resolutions, along with a list of application servers or clusters where the EJB JAR runs.

The product creates a composition unit from only one asset. However, multiple composition units can share a single asset. This is particularly useful in scenarios where different configurations use the same application binary files to provide different runtime behavior.

The following rules apply to a composition unit:

- A composition unit can exist only in a business-level application.
- Because a composition unit contains application-specific configuration and wiring information, multiple business-level applications cannot share an asset or shared library composition unit.

The following graphic shows the use of composition units in business-level applications. Assume that you have unprocessed files, such as archives, that you want to use in business-level applications. Before you can add the files to business-level applications, you must first import the files as assets, which adds the files to the product repository. Next, you add the assets to business-level applications, which creates composition units for the assets. Business-level applications can contain asset composition units, shared library composition units, or business-level composition units.



Importing assets

You must register application business logic such as Java Platform, Enterprise Edition (Java EE) archives, libraries, and other resource files with the product configuration as assets before you can add the assets to one or more business-level applications. Importing an asset registers it with the product configuration.

Before you begin

This topic assumes that you have one or more application binary files that you want to add to a business-level application. You must register those binary files as assets before you can add them to the business-level application.

About this task

Before a business-level application that uses an asset can be started on the target run time, the asset binaries must be extracted to a deployer-defined location on the file system that is local to the target run time. Importing an asset extracts binaries to a location that is local to the target run time.

The application server run time that reads the asset binaries either at application start time or while serving an incoming client request determines the extraction format of the asset binaries. The extraction format might include unzipping of Java archive (JAR) or compressed (zip) files.

This topic describes how to import an asset using the administrative console. Alternatively, you can use the wsadmin tool or programming.

Procedure

1. Click **Applications > New Application > New Asset** in the console navigation tree.
2. On the Upload asset page, specify the asset package to import.
 - a. Specify the full path name of the asset.
 - b. Click **Next**.
3. On the Select options for importing an asset page, specify asset settings.

You typically can click **Next** and use the default values.

 - a. Optional: For **Asset description**, specify a brief description of the asset.
 - b. Optional: For **Asset binaries destination URL**, specify the target location of the asset.

This setting specifies the location to which the product extracts the asset. After an asset is imported, the product looks for the asset in this location when a running application uses the asset. If you do not specify a value, the product installs the asset to the default location, `${profile_root}/installedAssets/asset_name/BASE/`.
 - c. Optional: For **Asset type aspects**, examine the asset content type and version specified by the product. You cannot change this setting value.

The type aspect typically denotes the type of application contents, such as a specification to which the application is written. For example, an enterprise bean (EJB) that supports the EJB Version 2.0 specification has the aspects `type=EJB,version=2.0`.

If the type aspect is none and if the asset is a JAR file, then the product associates a `javarchive` type aspect with the asset by default.
 - d. For **File permissions**, specify any file permissions that are set on asset binary files so the target run time can read or run the asset. Importing the asset extracts its binary files on the disk local to the target runtime environment.

Try importing the asset using the default value. For detailed information on the **File permissions** setting, refer to the Select options for importing an asset page online help.

Restriction: OSGi applications do not use a **File permissions** setting.

 - e. For **Current asset relationships**, add assets that the asset you are importing needs to run or remove assets that are not needed.

When the product imports a JAR asset, the product detects asset relationships automatically by matching the dependencies defined in the JAR manifest with the assets that are already imported into the administrative domain.

- f. For **Validate asset**, specify whether the product validates the asset.

The setting is deselected by default. This **false (no)** value is appropriate for most assets. Only select **true (yes)** to validate an asset when needed.

The product does not save the value specified for **Validate asset**. Thus, if you select to validate the asset (**yes**) now and later update the asset, when you update the asset you must enable this setting again for the product to validate the updated files.

Restriction: OSGi applications do not use a **Validate asset** setting.

- g. Click **Next**.

4. On the Summary page, click **Finish**.

Results

Several messages are displayed, indicating whether your asset is imported successfully.

An asset can contain multiple deployable objects as defined by the application contents of that asset. A *deployable object* is a part of the asset that you can map to a deployment target such as an application server or a cluster. If the product imports the asset successfully, then appropriate deployable objects are identified in the asset and are further used when a composition unit is created from that asset.

If the asset importing is not successful, read the messages and try importing the asset again. Correct the values noted in the messages.

What to do next

If the product imports the asset successfully and displays the list of assets on the Assets page, then click **Save**.

Add a composition unit to a business-level application using the asset that you imported. An asset included in a business-level application is represented by a composition unit.

Upload asset settings

Use this page to specify the asset to register with the asset repository. You can add registered assets to a business-level application.

To view this administrative console page, click **Applications > New application > New Asset**.

Importing an asset registers the asset with the asset repository.

The product manages the contents of a registered asset as a single entity. The contents of a registered asset must be accessible to application servers, web servers and other runtime environments that use the asset.

During asset importing, asset files typically are uploaded from a client workstation running the browser to the server running the administrative console, where they are registered. In such cases, use the web browser running the administrative console to select files to upload to the server.

Path to the asset

Specifies the fully qualified path to the asset.

Specify one of the following supported assets:

- A single file, such as an enterprise bean (EJB) file
- An archive of files, such as a Java archive (JAR) or a compressed .zip file
- An archive of archives, such as an enterprise archive (EAR) or shared library file

Use **Local file system** if the browser and asset files are on the same machine (whether or not the server is on that machine, too).

Use **Remote file system** if the asset file resides on any node in the current cell context. Only supported assets are shown during the browsing. Also use **Remote file system** to specify an asset file that is already residing on the machine running the application server. For example, the field value might be *profile_root/installableApps/my_bean.ejb*. After the asset file is transferred, the **Remote file system** value shows the path of the temporary location on the server.

Asset settings

Use this page to specify options for the registration of an asset with the asset repository. Default values for the options are used if you do not specify a value. If the asset is an OSGi application, additional information about bundle download status is displayed.

To view this administrative console page, click **Applications > Application Types > Assets > asset_name**. This page is similar to the Select options for importing an asset page on the asset import and update wizards.

Asset name

Specifies a logical name for the asset. An asset name must be unique within a cell and cannot contain an unsupported character.

An asset name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Table 29. Characters that you cannot use in a name. The product does not support these characters in a name.

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

This **Asset name** field is the same as the **Name** setting on an Assets page.

Information	Value
Data type	String

Asset description

Specifies a description for the asset.

Asset binaries destination URL

Specifies the directory to which the product imports the asset file.

Information	Value
Data type	String
Units	Full path name

Asset type aspects

Specifies the type of asset content. Examples of asset type include Java archive (JAR) files, shared libraries, enterprise application archive (EAR) files, and enterprise bundle archive (EBA) files.

The asset type suggests the content of the asset. An asset packaged as a JAR file might contain a web module, portlet, or web service. An asset packaged as an EBA file contains an OSGi application.

This setting is read-only. You cannot edit this setting.

Information	Value
Data type	String
Units	File type
Default	none

File permissions

Specifies access permissions for asset binaries that the product expands to the asset binaries destination URL.

Restriction: OSGi applications do not use a **File permissions** setting.

You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the list. List selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the list. Selecting multiple options combines the file permission strings.

Table 30. File permission string sets for list options. Select a list option or specify a file permission string in the text field.

Multiple-selection list option	File permission string set
Allow all files to be read but not written to	.*=755
Allow executables to execute	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Allow HTML and image files to be read by everyone	.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755

Instead of using the multiple-selection list to specify file permissions, you can specify a file permission string in the text field. File permissions use a string that has the following format:

file_name_pattern=permission#file_name_pattern=permission

where *file_name_pattern* is a regular expression file name filter (for example, *.*\\.jsp* for all JSP files), *permission* provides the file access control lists (ACLs), and # is the separator between multiple entries of *file_name_pattern* and *permission*. If # is a character in a *file_name_pattern* string, use \# instead.

If multiple file name patterns and file permissions in the string match a uniform resource identifier (URI) within the asset, then the product uses the most stringent applicable file permission for the file. For example, if the file permission string is *.*\\.jsp=775#a.*\\.jsp=754*, then the *abc.jsp* file has file permission 754.

Tip: Using regular expressions for file matching pattern compares an entire string URI against the specified file permission pattern. You must provide more precise matching patterns using regular expressions as defined by Java programming API. For example, suppose the product processes the following directory and file URIs during a file permission operation:

Table 31. Example URIs for file permission operations. Results are shown following this table.

Number	Example URL
1	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war
2	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
3	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF/MANIFEST.MF
4	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/WEB-INF/classes/MyClass.class
5	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/mydir/MyClass2.class
6	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF

The file pattern matching results are:

- MyWarModule.war does not match any of the URIs
- .*MyWarModule.war.* matches all URIs
- .*MyWarModule.war\$ matches only URI 1
- .*\\.jsp=755 matches only URI 2
- .*META-INF.* matches URIs 3 and 6
- .*MyWarModule.war/.*/.*\\.class matches URIs 4 and 5

If you specify a directory name pattern for **File permissions**, then the directory permission is set based on the value specified. Otherwise, the **File permissions** value set on the directory is the same as its parent.

For example, suppose you have the following file and directory structure:

```
/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
```

and you specify the following file pattern string:

```
.*MyApp.ear$=755#.*\\.jsp=644
```

The file pattern matching results are:

- Directory MyApp.ear is set to 755
- Directory MyWarModule.war is set to 755
- Directory MyWarModule.war is set to 755

Important: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

Access permissions specified here are at the asset level. You can also specify access permissions for asset binaries in the node-level configuration. The node-level file permissions specify the maximum (most lenient) permissions that can be given to asset binaries. Access permissions specified here at asset level can only be the same as or more restrictive than those specified at the node level.

Information	Value
Data type	String

Current asset relationships

Specifies the assets to which this asset is related.

To add or remove a relationship, use the Manage relationships page:

1. Click **Manage Relationships**. The **Selected** list on the right lists the current asset relationships.
2. To add a relationship, select an asset in the **Available** list on the left and click >>.
3. To remove a relationship, select an asset in the **Selected** list on the right and click <<.
4. Click **OK**.

Information	Value
Data type	String
Default	none

Validate asset

Specifies whether the product examines the asset references specified during asset importing or updating and, if validation is enabled, warns you of incorrect references or fails the operation.

Restriction: OSGi applications do not use a **Validate asset** setting.

An asset typically refers to resources using data sources for container-managed persistence (CMP) beans or using resource references or resource environment references defined in deployment descriptors. The validation checks whether the resource referred to by the asset is defined in the scope of the deployment target of that asset.

Select `true` (enable the check box) for resource validation and to stop operations that fail as a result of incorrect resource references. Select `false` (empty check box) for no resource validation.

Information	Value
Data type	String
Default	false (empty check box)

EBA Dependencies

For an enterprise bundle archive (EBA) asset, displays the current bundle download status for all bundles in the asset. This item is only displayed if your asset is an EBA asset, which means that it contains an OSGi application.

You cannot update an EBA asset until bundle downloads are complete from any previous update, and until the business-level application that uses the asset has picked up the previous updates by being restarted. Before you try and update bundle versions, you can use the EBA dependency information to check the bundle download status of the asset. The status displayed is one of the following values:

- Bundles downloading...
- Bundle downloads are complete.
- No bundles downloads are required.

Note: In addition to the information given here, you can also check the bundle download status indirectly, by checking the status of the associated EBA composition unit as described in Checking and updating the EBA asset version used by a business-level application.

If bundle downloads for the asset are complete, or no bundle downloads are required, you can update the asset using either of the methods described in Maintaining bundle versions for an EBA asset.

If bundle downloads for the asset are complete, and a new version of the EBA asset is available, restart the business-level application to bring the EBA composition unit up-to-date and to run the newer configuration.

Managing assets

After application binary files are imported and registered with the product management domain as assets, you can view, update and export those assets.

Before you begin

Import one or more assets. The name of each imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can view the contents of assets, update assets, remove assets from the product management domain, or export copies of assets to a target location. This topic describes how to perform these asset management operations from the administrative console Assets page. Alternatively, you can use programming or the wsadmin tool.

Procedure

- View or edit asset settings.
 1. Go to the administrative console Assets page.
Click **Applications > Application Types > Assets**.
 2. Click the asset name in the list of assets. The Asset settings page displays the values that are specified for the asset.
 3. Optional: Change the asset settings as needed and click **OK** to save the changes.
- Remove one or more assets from the product management domain.
- Update the contents of an asset.
- Export an asset to a target location.

What to do next

Create a business level application and add the asset to the business-level application.

Asset collection

Use this page to view a list of assets in the asset repository and to manage those assets. After importing an asset, you can add the asset to a business-level application.

Assets include Java archive (JAR) and compressed files that are used by applications installed on a server.

To view this administrative console page, click **Applications > Application Types > Assets**.

To view the values specified for an asset, click the asset name in the list. The displayed asset settings page shows the values specified. On the settings page, you can change existing asset values.

To manage an asset, enable the **Select** check box beside the asset name in the list and click a button:

Table 32. Button descriptions. Use the buttons to manage assets.

Button	Resulting action
Import	Opens a wizard that helps you add an asset to the asset repository.
Delete	Removes the asset from the asset repository and deletes the asset binaries from the file system of all nodes where the assets are installed. On single-server installations, deletion occurs after the configuration is saved. On multiple-server installations, deletion occurs after the configuration is saved and synchronized with the nodes.
Update	Opens a wizard that helps you update asset files. You can replace a file or module that exists on the server with a file or module that has the same name. Or you can add a new file or module, provided the new file or module does not have the same name as an asset that already exists on the server.
Export	Accesses the Export asset page, which you use to export an asset to a file at a location of your choice. Use the Export action to back up an asset.

Name

Specifies the name of the asset. Asset names must be unique within a cell and cannot contain an unsupported character.

Description

Specifies a description for the asset.

Updating assets

You can use the Update asset wizard to update classes, composites, wsdl, xsd, and definitions.xml files in an asset.

Before you begin

Import one or more assets. The file name of each deployable object in the imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can update all or part of the contents of assets that are in the product management domain. Complete the steps in the Procedure to update an asset using the administrative console Update asset wizard. Alternatively, you can update assets using programming or the wsadmin tool.

The following update limitations exist if the asset you are updating is a Service Component Architecture (SCA) asset:

- You cannot delete a composite file that a composition unit is using. If a delete is attempted, a warning message is sent to the Update asset log.
- You cannot update an sca-contribution.xml file.
 - SCA cannot detect deployable composites that are either added or deleted. Therefore, during deployment of a new composition unit, you do not see the new deployable composite in the deployables option list.
 - SCA cannot detect dependencies that are added/removed during the Update asset process.
 - If a new import package is added and if a class in an existing composition unit is updated to require this new package, then the Update asset wizard fails with a `ClassNotFoundException`. Deployment of any new composition units from the updated asset are successful as the dependencies are detected during deployment operation.
 - If a new export package/namespace is added, then it has no affect on the existing composition unit and the Update asset wizard completes successfully.
- You cannot update configuration metadata such as composite, XSD, or WSDL files in a dependent SCA asset that is deployed as a shared library. You can only update binary files that do not contain annotations which the SCA programming model uses or depends on.
- Because the Update asset wizard uses the new composite definition file provided in the asset for the existing composition unit, the following post deployment related changes to the composite configuration are not saved.
 - Binding resources: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.
 - Component reference target URIs: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.
 - Component properties: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.
 - HTTP Endpoint URL information: You need to reconfigure this information after the Update asset wizard finishes.
- For web services policy set attachments, during Update asset processing:

- If there is a policy set specified for an endpoint in the updated composite definition file, SCA checks to see if a policy set has already been attached to that endpoint in the deployed composition unit. If an attachment already exists for that endpoint, the attachment is removed, and the policy set listed in the new composite file for that endpoint is attached. In this situation, if you have made any post deployment policy set configuration changes, these changes are lost.
- If there is no policy set defined for an endpoint in the update composite definition file, then any existing attachments to that endpoint are removed.

Policy set bindings follow these same rules.

- For RunAs and RoleToUser mapping definitions, during Update asset processing:
 - For implementation.java, implementation.spring and implementation.osgiapp, any new roles defined in the definition.xml file in the asset are picked up and users can be mapped to these roles using either the editCompositionUnit command or the administrative console. Any existing role mappings for the original roles are preserved.
 - For implementation.jee, the runAs and RoleToUser mappings are defined in the JEE application instead of in the SCA asset or SCA composition unit. Therefore, SCA does not do anything with these mappings during Update asset processing.
- The user defined virtual host that hosts web content for binding.ws, binding.atom, binding.http with wireformat.jsonrpc and implementation.widget is not supported. A virtual host mapping of default_host is used during Update asset processing.

Procedure

1. Go to the Update asset wizard.
 - a. Click **Applications > Application Types > Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to update.
 - c. Click **Update**.
2. On the Update asset page, specify whether you want replace an entire asset or update its contents and, as needed, the replacement file or module.
 - a. Select an update option.

You can update asset contents by adding, deleting, or updating a single file or module in the asset, or by merging multiple files or modules. Update options include the following:

 - Replace entire asset
 - Replace specific asset contents
 - Add module or file to asset
 - Remove file or module from asset
 - Merge asset contents

The online help for the Update asset page describes the options.
 - b. If you are updating specific asset contents or removing a file or module, specify the path beginning with the asset archive file.

For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at com/company/greeting.class in module hello.jar, specify a relative path of hello.jar/com/company/greeting.class.
 - c. If you are updating the entire asset, updating an asset file or module, or merging asset contents, specify the full path name of the new file or module.
 - d. Click **Next**.
3. On the Select options for updating an asset page, specify asset settings and click **Next**.

The online help for the Select options for importing an asset page describes the settings.
4. On the Summary page, click **Finish**.

Results

If you update an asset packaged as a library JAR file that is not a Java Platform, Enterprise Edition (Java EE) archive, then the product automatically distributes the updated asset to all of the composition units that use the asset.

However, if you update a Java EE asset, then the product does not automatically distribute the updated Java EE archive to composition units created from that asset, which are Java EE applications. You must select every Java EE application created from that asset and use the **Update** button to update the Java EE application individually by specifying the update contents.

What to do next

Create a business-level application and add the asset to the business-level application.

Update asset settings

Use this page to select whether you want replace an entire asset or update its contents. You can update asset contents by adding, deleting, or updating a single file or module in the asset, or by merging multiple files or modules into an asset. Updating an asset registers the updated files with the product management domain.

To view this administrative console page, click **Applications > Application Types > Assets**, select the asset to update, and then click **Update**.

The product manages the contents of a registered asset as a single entity. The contents of a registered asset must be accessible to application servers, web servers and other runtime environments that use the asset.

When you replace an asset or update an asset by adding a file or module, asset files typically are uploaded from a client workstation running the browser to the server machine running the administrative console, where they are registered. In such cases, use the web browser running the administrative console to select files to upload to the server machine.

The specified asset that you are installing must be one of the following supported assets:

- A single file, such as an enterprise bean (EJB) file
- An archive of files, such as a Java archive (JAR) or a compressed .zip file
- An archive of archives, such as an enterprise archive (EAR) or shared library file

Replace entire asset:

Under **Select the type of update to perform**, specifies to replace the entire asset installed on the server with a new (updated) asset.

After selecting this option, specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.

Use **Local file system** if the browser and asset files are on the same machine (whether or not the server is on that machine, too).

Use **Remote file system** if the asset file resides on any node in the current cell context. Only supported assets are shown during the browsing. Also use **Remote file system** to specify an asset file that is already residing on the machine running the application server. For example, the field value might be `profile_root/installableApps/my_bean.ejb`. After the asset file is transferred, the **Remote file system** value shows the path of the temporary location on the server.

Replace specific asset contents:

Under **Select the type of update to perform**, specifies to replace a file or module of the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.
3. Click **Next**.

The **Replace entire asset** description describes options for specifying the full path name of an asset or file to add using **Local file system** and **Remote file system** options.

Add a module or file to an asset:

Under **Select the type of update to perform**, specifies to add a file to the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.

The **Replace entire asset** description describes options for specifying the full path name of an asset or file to add using **Local file system** and **Remote file system** options.

Remove a file or module from an asset:

Under **Select the type of update to perform**, specifies to remove a file or module from the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file to be removed that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Click **Next**.

Merge asset contents:

Under **Select the type of update to perform**, specifies to compare the new file or module with the file or module of the asset installed on the server. If the file or module exists, it is replaced. Otherwise, it is added to the installed asset.

After selecting this option, specify whether the new file or module is on a local or remote file system and the full path name of the file or module. The path provides the location of the updated asset before installation.

The **Replace entire asset** description describes options for specifying the full path name of a file or module to merge using **Local file system** and **Remote file system** options.

Update associated composition unit:

Specifies whether to update the composition units that are associated with an enterprise (Java EE) asset. This option applies to enterprise assets only.

The default value is NONE. Specify ALL to update all of the composition units that are associated with the enterprise asset.

Deleting assets

You can remove application binary files that are registered as assets from the product management domain.

Before you begin

Import one or more assets. The name of each imported asset is shown on the list of assets on the administrative console Assets page.

About this task

You can remove assets from the product management domain, provided the asset does not have an existing composition unit. If an asset has one or more composition units defined in the management domain, then you cannot delete that asset until those composition units are removed.

This topic describes how to delete assets using the administrative console. Alternatively, you can use programming or the wsadmin tool.

Procedure

1. Go to the Delete asset page.
 - a. Click **Applications** > **Application Types** > **Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to delete.
 - c. Click **Delete**.
2. On the Delete asset page, click **OK** to confirm that you want the specified asset removed from the product management domain.

Click **Cancel** to return to the Assets page and not delete the asset.

Results

The product deletes the asset from the product management domain.

In a multiple-server environment, the product deletes the asset binary files from the target node machine after node synchronization.

What to do next

On the Assets page, verify that the deleted asset is no longer in the list of imported assets.

Exporting assets

After application binary files are imported and registered with the product management domain as assets, you can export those assets.

Before you begin

Import one or more assets. The file name of each deployable object in the imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can export copies of assets to a target location. Exporting stores application binary files, enabling you to back up the files or edit them. The file resulting from exporting an asset contains configuration information for the asset.

This topic describes how to export an asset from the administrative console Assets page. Alternatively, you can use programming or the wsadmin tool.

Procedure

1. Go to the Export asset page.
 - a. Click **Applications > Application Types > Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to export.
 - c. Click **Export**.
2. On the Export asset page, click the asset name or identifier.

To cancel the export operation and return to the Assets page, click **Back**.
3. Specify the target location for the asset file.

What to do next

Examine the target file to verify that the asset exported correctly. You can later edit this file and import the edited asset.

Creating business-level applications

You can create an empty business-level application and then add assets, shared libraries, business-level applications, and other artifacts as composition units to the empty business-level application.

Before you begin

Configure each target application server as needed. You must deploy a business-level application to a Version 7.0 or later server or cluster.

Optionally, determine what assets or other files that you want to add to your business-level application and whether your application files can run on your deployment targets.

About this task

You can create business-level applications using the administrative console, programming, or the wsadmin tool.

Procedure

1. Select a way to create your business level application.

Table 33. Ways to create business level applications. You can create business-level applications using the administrative console, programming, or wsadmin.

Option	Method
Administrative console business-level application creation wizard See "Creating business-level applications with the console" on page 217.	Click Applications > New application > New Business-level Application and follow instructions in the wizard.

Table 33. Ways to create business level applications (continued). You can create business-level applications using the administrative console, programming, or wsadmin.

Option	Method
Administrative console Java Platform, Enterprise Edition (Java EE) application installation wizard See “Installing enterprise application files with the console” on page 126.	Click Applications > New application > New Enterprise Application and follow instructions in the wizard. The product creates a new business-level application with the enterprise application that you install or makes the enterprise application a composition unit of an existing business-level application. See the Business-level application name setting on the Select installation options wizard page.

2. Create your business-level application using the administrative console, programming or wsadmin.
3. Save the changes to your administrative configuration.
When saving the configuration, synchronize the configuration with the nodes where the application is expected to run.

Results

The name of the application is shown in the list on the Business-level applications page.

What to do next

After you create a business-level application, you can do the following to add composition units to it:

1. Import any assets needed by your business-level application.
2. Add assets, shared libraries, or other business-level applications as composition units.
3. Save the changes to your administrative configuration.
4. Start the business-level application.

If the application does not run as desired, edit the application configuration, then save and run it again.

Creating business-level applications with the console

You can create an empty business-level application and then add assets or business-level applications as composition units to the empty business-level application.

Before you begin

Before you create a business-level application, decide upon an application name. Optionally, determine which assets, shared libraries, or business-level applications that the new business-level application needs.

About this task

This topic describes how to create an empty business-level application and then add assets as composition units to the application using the administrative console. Alternatively, you can use programming or the wsadmin tool.

You can add an asset or shared library composition unit to multiple business-level applications. However, each composition unit for the same asset must have a unique composition unit name. You can add a business-level application composition unit to more than one business-level application.

Procedure

1. Create an empty business-level application.
 - a. Click **Applications > New application > New Business Level Application**.

- b. On the New business-level application page, specify a unique name for the application and a description, and then click **OK**.
- c. On the business-level application settings page, click **Save**.

The name and description are shown in the list of applications on the Business-level applications page. Because the application is empty, its status is `Unavailable`.

2. Optional: Add one or more assets, non-Java EE shared libraries, or business-level applications to a business-level application.

The product adds these assets as composition units of your business-level application.

- If the asset that you want to add to your business-level application is a Java Platform, Enterprise Edition (Java EE) application or module that is not yet deployed, see step 3.
- If the asset is a Java EE shared library, see step 4.

- a. Import the assets or create the business-level applications that you want to add to the business-level application.

- b. Go to the business-level application settings page.

Click **Applications > Application Types > Business-level applications > *application_name***.

- c. On the business-level application settings page, specify the type of composition unit to add.

- To add an asset, under **Deployed assets**, click **Add > Add Asset**.
- To add a shared library, under **Deployed assets**, click **Add > Add Shared Library**.
- To add a business-level application, under **Business-level applications**, click **Add**.

- d. On the Add page, select a unit from the list of available units, and then click **Continue**.

If you are adding one or more deployable unit assets and you have multiple imported assets available, you can select more than one deployable unit.

- e. On the Set options page, change the composition unit settings as needed, and then click **Next**.

This page is not shown when you add a Java EE asset as a shared library or if you have multiple deployable unit assets. If the application installation or update wizard displays and you want to add a Java EE asset as a shared library, see step 4.

- f. On the Map composition unit to a target page, change the deployment target as needed, and then click **Next**.

This page is not shown when you add a business-level application.

- g. If you are adding one or more deployable unit assets, specify composition unit relationship options. See “Relationship options settings” on page 226.

- h. On the Summary page, click **Finish**. Several messages are displayed, indicating whether the product adds the unit to the business-level application successfully. A message having the format `Completed res=[WebSphere:cuname=unit_name,cuedition=version]` indicates that the addition is successful. Click **Manage application**.

If the product adds the unit successfully, the name of the unit is shown on the list of composition units on the Adding composition unit to the business-level application page.

If the unit addition is not successful, read the messages and try adding the unit again. Correct the problems noted in the messages.

- i. On the Adding composition unit to the business-level application page, click **Save**.

The product creates composition units for the asset, shared library, or business-level application. The unit names are shown in lists of composition units on the settings page of your business-level application. To view the settings page, click **Applications > Application Types > Business-level applications > *your_application_name***.

3. Optional: Install a Java EE application or module, and add it as a composition unit to your business-level application.

When installing an enterprise archive (EAR) file or a stand-alone Java EE module using the application installation wizard, you can specify a business-level application to which to add the EAR file or module.

You can also specify relationships to any shared libraries that your Java EE application or module uses. The product creates composition units that represent those relationships.

- a. Click **Applications > New application > New Enterprise Application**.
- b. On the first Preparing for the application installation page, specify the Java EE application or module to install and click **Next**.
- c. On the second Preparing for the application installation page, select **Detailed - Show all installation options and parameters**, specify whether to generate default bindings and mappings as needed for the application or module, and click **Next**.
- d. On the Select installation options page of the wizard, select your business-level application for **Business-level application name** and click **Next**. The product creates a composition unit that has the same name as the Java EE application or module and adds the unit to your business-level application.

If you do not specify a value for **Business-level application name**, then the product creates a default business-level application that has the same name as the Java EE application that you are installing. The product does not add the Java EE application as a composition unit to the business-level application that you created in step 1.

- e. Optional: On the Map shared library relationship page of the wizard, specify relationship identifiers and composition unit names for shared libraries that modules in your Java EE application use. The product creates a composition unit for each shared library relationship in your business-level application.

You can map shared library relationships when installing your Java EE application or module or, after installation, return to the Map shared library relationship page and specify shared library relationships. See step 4.

- f. Complete the other application installation wizard options as needed to install the Java EE application or module.

The product creates composition units for the application, module, or shared library relationships. The unit names are shown in lists of composition units on the settings page of your business-level application. To view the settings page, click **Applications > Application Types > Business-level applications > your_application_name**.

4. Optional: After installation of a Java EE application or module, you can specify composition units for relationships to shared libraries that are used by your business-level application. Specify relationships to shared libraries on the Map shared library relationship page of the application installation or update wizard.

- a. If you have not done so already, import a Java EE asset such as an enterprise bean (EJB) or web module (WAR) that uses a shared library file.

If the product displays `javaarchive` for **Asset type aspects** on the asset settings page, continue to step 4b.

If the product does not display `javaarchive` for **Asset type aspects** on the asset settings page, then the asset is not a Java EE asset. Use step 2 to add a shared library to your business-level application.

- b. Go to a settings page for your business-level application.
Click **Applications > Application Types > Business-level applications > your_application_name**.
- c. Under **Deployed assets**, click **Add > Add Shared Library**.
- d. On the Add composition unit page, select the Java EE asset that you imported and then click **Continue**.

The Java EE application installation or update wizard displays. Select the Java EE application or module that uses the asset, and complete the steps in the wizard.

- e. On the Select installation options page of the wizard, select your business-level application for **Business-level application name**.

- f. On the Map shared library relationship page of the wizard, specify a relationship identifier and composition unit name for the asset.
- g. Complete the other wizard options as needed.

The product creates a composition unit for the shared library relationship. The unit name is shown in the list of deployed asset composition units on the settings page of your business-level application.

Results

The name of your business-level application is shown on the Business-level applications page in the list of applications.

What to do next

After you create the application, save the changes to your configuration and start the application as needed.

Business-level application collection

Use this page to view and manage business-level applications.

To view this administrative console page, click **Applications > Application Types > Business-level applications**.

To view the values specified for an application configuration, click the application name in the list. The displayed application settings page shows the values specified. On the settings page, you can change existing configuration values and link to additional console pages that assist you in configuring the application.

To manage a business-level application, enable the **Select** check box beside the application name in the list and click a button:

Table 34. Button descriptions. Use the buttons to manage business-level applications.

Button	Resulting action
Start	Attempts to run the application. After the application starts successfully, the state of the application changes to <i>Started</i> if the application starts on all deployment targets, else the state changes to <i>Partial Start</i> .
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to <i>Stopped</i> if the application stops on all deployment targets, else the state changes to <i>Partial Stop</i> .
New	Opens a wizard that helps you add assets, shared libraries, or business-level applications as composition units to your application.
Delete	Deletes the application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed. On single-server installations, deletion occurs after the configuration is saved. On multiple-server installations, deletion occurs after the configuration is saved and synchronized with the nodes.

Name:

Specifies the name of the business-level application. Application names must be unique within a cell and cannot contain an unsupported character.

Description:

Specifies a description for the business-level application.

Status:

Indicates whether the application deployed on the application server is started, stopped, or unknown.

Table 35. Application status. The status indicates whether the application is running.

Icon	Application status	Description
	Started	Application is running.
	Partial start	Application is in the process of changing from a <i>Stopped</i> state to a <i>Started</i> state. Application is starting to run but is not fully running yet. Or, it cannot fully start because a server mapped to one or more application modules is stopped.
	Stopped	Application is not running.
	Partial stop	Application is in the process of changing from a <i>Started</i> state to a <i>Stopped</i> state. Application has not stopped running yet.
	Unknown	Status cannot be determined.
	Pending	Status is temporarily unknown pending an event that a user did not initiate, such as pending an asynchronous call.
	Not applicable	Application does not provide information as to whether it is running.

The status of an application on a web server is always **Unknown**.

New business-level application settings

Use this page to name and describe a new business-level application.

To view this administrative console page, click **Applications > New application > New Business-level Application**.

Name:

Specifies a logical name for the business-level application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Table 36. Characters that you cannot use in a name. The product does not support these characters in a name.

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Information

Data type

Value

String

Description:

Specifies a description for the application.

This field is the same as the **Description** setting on a Business-level applications page.

Shared library relationship and mapping settings

Use the Shared library relationship and Shared library relationship mapping pages to specify relationship identifiers and composition unit names for shared libraries that modules in your enterprise application reference. When installing your enterprise application, the product creates a composition unit for each shared library relationship in the business-level application that you specified on the Select installation options page of the application installation wizard.

To view this console page in a wizard, click **Applications > Install new application > New Enterprise Application > *application_path* > Next > Detailed - Show all installation options and parameters > Next > *application_name* > Step: Map shared library relationships.**

After installation, click **Applications > Application Types > WebSphere enterprise applications > Shared library relationships.**

To map library files used in a business-level application to an application or web module, use the Shared library relationship mapping page:

1. Click **Reference shared libraries.**
2. Note the application or module in **Map libraries to the application or module listed.** You are associating library files with that application or module.
3. From the **Available** list, select one or more libraries that the application or module uses.
4. Click >> to add them to the **Selected** list.
5. To remove an association, select one or more libraries in the **Selected** list and click <<.
6. Click **OK.**

Module:

Specifies the name of the module associated with the shared libraries.

URI:

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

Relationship identifiers:

Specifies an identifier for a module shared library relationship. The product assigns an identifier to the composition unit that it creates for the shared library relationship in the business-level application.

Composition unit names:

Specifies a composition unit name for the shared library relationship. The product uses this value to name the composition unit that it creates for the shared library relationship in the business-level application that you specified on the Select installation options page of this wizard.

This setting is only in the application installation and update wizards.

Match target:

Specifies whether the product maps the composition unit for the shared library relationship to the same deployment target as the business-level application.

Note: If you later change the deployment target of the business-level application or its modules, you must manually update the shared library target to match the target of the application and modules. The targets of shared library composition units are not automatically updated. Not updating the target of the shared library composition unit might cause `java.lang.ClassNotFoundException` errors and prevent the application or its modules from starting. To prevent these error conditions, also ensure that shared libraries upon which other modules or applications depend have a lower starting weight than dependent applications and modules.

Add composition unit settings

Use this page to specify options for the composition unit to be added to the business-level application. The product assigns a default value for an option when you do not specify a value.

To view this administrative console page, click **Applications > Application Types > Business-level applications > *business-level_application_name* > Add > Add unit_type**.

Name:

Specifies the name of the composition unit to be added to the business-level application.

The table lists available composition units. Select a unit from this list.

Description:

Specifies a description for the composition unit.

Add asset settings

Use this page to add one or more assets to a business-level application.

To view this administrative console page, click **Applications > Application Types > Business-level applications > *application_name* > Add > Add Asset**.

Deployable units:

Specifies the imported assets available for use in a business-level application. The list of deployable units includes only imported assets, and not shared libraries or business-level applications.

From this list, select one or more deployable units to add as composition units to your business-level application.

Set options settings

Use this page to specify options for the composition unit to be added to the business-level application. The product supplies default values for the options if you do not specify a value.

To view this administrative console page, click **Applications > Application Types > Business-level applications > *application_name***. On the business-level application settings page, specify the type of composition unit to add:

- To add an asset, under **Deployed assets**, click **Add > Add Asset**.
- To add a shared library, under **Deployed assets**, click **Add > Add Shared Library**.
- To add a business-level application, under **Business-level applications**, click **Add**.

Backing identifier:

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the format: `WebSphere:unit_typename=unit_name,unit_typeversion=version_number`. For example, for the `MyApp.jar` asset, the backing identifier might be `WebSphere:assetname=MyApp.jar`.

Information	Value
Data type	String
Units	Composition unit identifier

Name:

Specifies the name of the composition unit.

For example, for the `MyApp.jar` asset, the name might be `MyApp.jar`.

A unit name cannot begin with a period (`.`), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Table 37. Characters that you cannot use in a name. The product does not support these characters in a name.

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Information	Value
Data type	String

Description:

Specifies a description for the composition unit.

Starting weight:

Specifies the order in which composition units are started when the server starts. The starting weight is like the startup order. The composition unit with the lowest starting weight is started first.

The value that you set for **Starting weight** determines the importance or weight of a composition unit within the business-level application. For example, for the most important composition unit within a business-level application, specify 1 for **Starting weight**. For the next most important composition unit within the business-level application, specify 2 for **Starting weight**, and so on.

Information	Value
Data type	Integer
Default	1
Range	0 to 2147483647

Start composition unit upon distribution:

Specifies whether to start the composition unit after the product distributes the composition unit to other locations.

The default is not to start the composition unit.

Information	Value
Data type	Boolean
Default	false

Restart behavior on update:

Specifies whether the product restarts deployment targets after updates to the composition unit.

Usually, a composition unit is mapped to one or more deployment targets. This setting determines whether the product restarts those targets after editing the composition unit.

Table 38. Restart behavior on update options. Depending on your selection, the product restarts all target nodes, the nodes controlled by sync plug-ins, or no nodes.

Option	Description
ALL	The product restarts each target node of the composition unit after editing the composition unit.
DEFAULT	The product restarts the nodes controlled by the sync plug-ins after editing the composition unit.
NONE	The product does not restart nodes after editing the composition unit.

Map target settings

Use this page to map a composition unit to a deployment target. The product assigns a default target when you do not specify a target.

To view this administrative console page, click **Applications > Application Types > Business-level applications > application_name > composition_unit_name > Modify Target**. The Map target page is similar to the Map composition unit to a target page in the add composition unit wizard.

On single-server products, a deployment target can be an application server or web server.

On multiple-server products, a deployment target can be an application server, cluster of application servers, or web server.

On this page, map a composition unit to one or more desired targets.

Current targets:

Specifies the existing deployment targets for the composition unit.

Available:

Lists the names of available deployment targets. This list is the same for every composition unit that is registered in the cell.

From this list, select only appropriate deployment targets for a composition unit.

If the unit calls a Version 8.5 application programming interface (API) or uses an 8.5 feature, then you must map the unit to an 8.5 deployment target. If the unit supports Java Platform, Enterprise Edition (Java EE) 6, then you must map the unit to an 8.0 or later deployment target.

If the unit calls a Version 7.x API, uses a 7.x feature, or supports Java EE 5, then you must map the unit to an 8.x or 7.x deployment target.

If the unit supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you must map the unit to an 8.x, 7.x or 6.x deployment target. You can map units that call a 6.x API or use a 6.x feature to an 8.x, 7.x or 6.x deployment target.

To map a composition unit to a deployment target, select a target from the **Available** list and click >>. The target name is displayed in the **Selected** list.

Selected:

Lists the names of desired deployment targets.

When you click **OK**, the product maps the composition unit to the deployment targets in the **Selected** list.

To remove a deployment target from the **Selected** list, select the target and click <<.

Relationship options settings

Use this page to specify relationship options for deployable or composition units in an asset deployed as part of a business-level application. Specifying a relationship declares a dependency relationship that a deployable unit or composition unit has on another asset deployed as a shared library in the same business-level application.

To view this administrative console page, click **Applications > Application Types > Business-level applications > application_name > deployed_asset_name > Relationship options**. This help also pertains to wizard pages that are shown when you add multiple deployable or composition unit assets to a business-level application. These pages are shown for the **Define relationship with existing composition units** and **Options for creating new composition units to satisfy asset relationships** wizard steps.

A business-level application consists of composition units. When you add an asset to a business-level application, the product creates a composition unit for the asset. The composition unit name can be different from the name of the asset being deployed. The list of deployed assets shown for a business-level application consists of the composition unit names for the deployed assets. The relationships defined in this page are composition unit relationships. The deployable units listed for a composition unit are those you chose from the associated asset when adding the asset. Composition unit relationships are expressed as deployable unit dependencies on other composition units belonging to the same business-level application. Only a composition unit for an asset deployed as a shared library can be specified as a dependency. You can map each deployable unit to a target independently from the others. Modifying relationships in this page only affects the composition unit, not the associated asset.

To specify relationship options, select a deployable unit and click a button.

Button	Resulting action
Set Relationships	Displays a page through which you can add or change relationships for the deployable unit. Specify a relationship if a deployable unit depends on another asset deployed as a shared library in order to run.
Enable Match Targets	This button is on the Set relationship options page. If the deployable unit has a dependency relationship defined, click Enable Match Targets to map the related deployed assets to the same deployment targets as the dependent deployable unit.
Disable Match Targets	If the deployable unit has a dependency relationship defined, click Disable Match Targets if the related deployed assets do not need to be deployed to the same targets as the deployable unit.

Deployable unit name or composition unit name:

Specifies the name of the deployable unit or the composition unit of the selected deployed asset.

Relationship:

Specifies the composition unit names for all relationships defined for the associated deployable unit.

This setting is on the Set relationship options page.

By default, a deployable unit has no relationships. To add or change related composition units, do the following:

1. Select the deployable unit.
2. Click **Set Relationships**.
3. Select the composition units that the deployable unit requires by moving them from the **Available** list to the **Selected** list.
4. Click **OK**.

Match targets:

Indicates the match targets value selected for the associated deployable unit. The default value is true.

A match targets value of true maps the composition units listed under **Relationship** to the same deployment targets as the associated deployable unit. Typically, you must deploy related composition units to the same targets as the dependent deployable unit in order for the deployable unit to run.

A false value indicates that the related composition unit can map to deployment targets which are different from the deployment targets of the deployable unit.

To set the value to true, select the deployable unit and click **Enable Match Targets**. To set the value to false, select the deployable unit and click **Disable Match Targets**. To set this value, the deployable unit must have a related composition unit.

If you have multiple deployable units and your security configuration supports multiple domains, the deployable units must be in the same security domain.

Business-level application settings

Use this page to configure a business-level application.

To view this administrative console page, click **Applications > Application Types > Business-level applications > application_name**.

This page is the same as the Adding composition unit to the business-level application page.

Name

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Table 39. Characters that you cannot use in a name. The product does not support these characters in a name.

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark

Table 39. Characters that you cannot use in a name (continued). The product does not support these characters in a name.

Unsupported characters		
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Information	Value
Data type	String

Description

Specifies a description for the business-level application.

Deployed assets

Specifies the asset and shared library composition units in the business-level application. A *composition unit* is a registered asset or shared library that has additional configuration information, which you specify when adding the asset to the application.

For each composition unit, the table provides a name, description, asset type, and the runtime status of the composition unit.

Table 40. Deployed assets button descriptions. Use the buttons to add or delete composition units.

Button	Resulting action
Add > Add Asset	For assets that contain Java Platform, Enterprise Edition (Java EE) applications or modules, opens the application installation wizard. On the Select installation options page of this wizard, you can specify a Business-level application name value that identifies the target business-level application. On the Map shared library relationships page, you can identify the shared library files that individual modules need to run and specify composition unit names for the module-shared library relationships. For non-Java EE assets, opens a wizard that helps you add an asset as a composition unit to your business-level application.
Add > Add Shared Library	Opens a wizard that helps you add a library file as a composition unit to your business-level application.
Delete	Deletes the composition unit from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed. On multiple-server installations, deletion occurs after the configuration is saved and synchronized with the nodes.

Business-level applications

Specifies the business-level applications in this business-level application.

The table provides a name, description, and the runtime status of each contained business-level application.

Table 41. Business-level applications button descriptions. Use the buttons to add or delete composition units.

Button	Resulting action
Add	Opens a wizard that helps you add a business-level application to your business-level application.
Delete	<p>Deletes the business-level application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed.</p> <p>On multiple-server installations, deletion occurs after the configuration is saved and synchronized with the nodes.</p>

Composition unit settings

Use this page to view composition unit settings and to change the configuration properties of a composition unit. The specific settings that are available for configuration can vary, depending upon the contents of the composition unit. For example, there are additional configuration settings if the asset contained in the composition unit is an SCA composite, or an OSGi application.

To view this administrative console page, click **Applications > Application Types > Business-level applications > application_name > deployed_asset_name**. The deployed asset is a composition unit of the business-level application.

- “Settings that are common to all composition units”
- “Additional composition unit settings for SCA composites” on page 231
- “Additional composition unit settings for OSGi applications” on page 231

Settings that are common to all composition units

Name:

Specifies a logical name for the composition unit. You cannot change the name on this page.

Description:

Specifies a description for the composition unit.

Backing ID:

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the format `WebSphere:unit_type=unit_name`. For example, for the `MyApp.jar` asset, the backing identifier might be `WebSphere:assetname=MyApp.jar`.

You cannot change the identifier on this page.

Information	Value
Data type	String
Units	Configuration unit identifier

Starting weight:

Specifies the order in which composition units are started when the server starts. The starting weight is like the startup order. The composition unit with the lowest starting weight is started first.

The value that you set for **Starting weight** determines the importance or weight of a composition unit within the business level application. For example, for the most important composition unit within a business-level application, specify 1 for **Starting weight**. For the next most important composition unit within the business-level application, specify 2 for **Starting weight**, and so on.

Note: Assign composition units upon which other composition units depend a lower starting weight than the dependent composition units. If a composition unit is not started and running before its dependent composition units, `java.lang.ClassNotFoundException` errors might result when you attempt to start the application or its modules.

Information	Value
Data type	Integer
Default	1
Range	0 to 2147483647

Start on distribution:

Specifies whether to start the composition unit when the product distributes the composition unit to other locations.

The default is not to start the composition unit.

This setting applies to asset or shared library composition units. This setting does not apply when the composition unit is a business-level application.

Information	Value
Data type	Boolean
Default	false

Recycle behavior on update:

Specifies whether the product restarts the composition unit after the composition unit is updated.

The default is to restart the composition unit after partial updating of the composition unit.

This setting applies to asset or shared library composition units. This setting does not apply when the composition unit is a business-level application.

Table 42. Option descriptions. Specifies whether to restart an asset or shared library composition unit.

Option	Description
ALL	Restarts the composition unit after the entire composition unit is updated
DEFAULT	Restarts the composition unit after the part of the composition unit is updated
NONE	Does not restart the composition unit after the composition unit is updated

Target mapping:

Specifies the current targets for the composition unit.

To change the deployment targets, click **Modify targets** then select a different set of deployment targets from the list of available clusters and servers.

For SCA, you must specify only a single server or cluster as the target. Do not map an SCA composition unit to multiple servers or clusters.

Note: When you change the deployment target of composition units in a business-level application, the startup order changes to the same order in which you remap composition unit targets, even if the starting weight for all composition units is set to 1. To avoid `java.lang.ClassNotFoundException` errors when attempting to start the remapped composition units, remap targets for composition units in the same order as that used to add the composition units or, after remapping, check starting weights to ensure that composition units upon which other composition units depend are started first.

Additional composition unit settings for SCA composites

SCA composite components:

Specifies the component names and component implementations of SCA composites in the application.

Table 43. Column descriptions. Provides the name of each component and the name of the class or code implementing the component.

Column	Description
Component Name	Specifies the name of a component associated with the SCA composite.
Component Implementation	Specifies the name of the class or code implementing the component.

None indicates that the SCA composite does not have defined components.

SCA composite properties:

Specifies the names and values of SCA composite properties in the application.

Table 44. Column descriptions. Provides the name and value of SCA composite properties.

Column	Description
Property Name	Specifies the name of an SCA composite property.
Property Value	Specifies the value of the property.

None indicates that the SCA composite does not have defined name-value properties.

SCA composite wires:

Specifies the sources and targets of wires in the SCA composite.

Table 45. Column descriptions. Provides the source and target of wires.

Column	Description
Wire Source	Specifies the source of a wire in the SCA composite.
Wire Target	Specifies the target of the wire.

None indicates that the SCA composite does not have defined wires.

Additional composition unit settings for OSGi applications

OSGi application deployment status:

The deployment status shows whether updates are available for the EBA asset that is contained in the composition unit. If a new version of an EBA asset is available, and all bundle downloads for the asset are complete, you can update the EBA composition unit so that the business-level application uses the latest configuration. You do not have to update the composition unit every time you update the asset.

There are four distinct deployment statuses for an EBA composition unit:

Using latest OSGi application deployment.

The composition unit is running the latest configuration of the backing asset and any CBA extensions.

New OSGi application deployment not yet available because it requires bundles that are still downloading.

The backing asset is currently undergoing a bundle version update, or bundles are downloading for a CBA extension.

New OSGi application deployment available.

The backing asset is available at a newer configuration than the configuration that is currently running in this composition unit, or a CBA extension has been added or replaced.

New OSGi application deployment cannot be applied because bundle downloads have failed.

The last bundle version update for the backing asset or CBA extension did not succeed, and therefore the newer configuration is not yet available.

If the status is “New OSGi application deployment available”, the **Update to latest deployment ...** button is available. Click this button to bring the EBA composition unit up-to-date and run the updated business-level application. If any of the updates need configuration changes, a wizard prompts you to update the configuration information.

When you save the changes to the EBA composition unit, the associated business-level application is updated to use the new configuration. If the business-level application is running, the bundle and configuration updates are applied immediately. If possible (that is, depending on the nature of the updates) the system applies the updates without restarting the application. Updates that pull in new use bundles at run time prompt a full restart of the application. Updates that pull in new provision bundles might also prompt a full application restart.

Example: Creating a business-level application

You can add many different types of artifacts to business-level applications. For example, you can add Java Platform, Enterprise Edition (Java EE) applications or modules, Java archives (JAR files), data in compressed files, and other business-level applications.

About this task

An example of creating a simple business-level application follows. This example assumes that you have a compressed file, such as a compressed file, or other archive available on your computer or on a remote server that you can use to complete the example.

If you do not have a compressed file available, look in product directories. Installing the product samples adds several sample files to the `/samples` directory. You can use these sample files in a business-level application.

Procedure

1. Import assets.
 - a. Click **Applications > New application > New Asset** in the console navigation tree.
 - b. On the Upload asset page, specify the asset package to import and click **Next**.
For example, specify a compressed file such as a compressed file and click **Next**.
 - c. On the Select options for importing an asset page, click **Next**.
 - d. On the Summary page, click **Finish**.
 - e. On the Adding asset to repository page, if messages show that the operation completed, click **Manage assets**.

- f. On the Assets page, click **Save**.
The file name displays in the list of assets.
2. Create an empty business-level application named MySampleBLA.
 - a. Click **Applications > New application > New Business Level Application**.
 - b. On the New business-level application page, specify a unique name such as MySampleBLA and a description, and then click **OK**.
 - c. On the business-level application settings page, click **Save**.
The name and description are shown in the list of applications on the Business-level applications page. Because the application is empty, its status is Unavailable.
3. Add the asset composition unit to your business-level application.
 - a. On the Business-level applications page, click the application name in the list of applications.
 - b. On the business-level application settings page, click **Add > Add Asset**.
 - c. On the Add composition unit page, select an asset composition unit from the list of available units, and then click **Continue**.
For example, select the compressed file asset and then click **Continue**.
 - d. On the Set options page, click **Next**.
 - e. On the Map composition unit to a target page, change the target server as needed, and then click **Next**.
 - f. On the Summary page, click **Finish**. Several messages are displayed. A message having the format `Completed res=[WebSphere:cuname=unit_name]` indicates that the addition is successful.
 - g. If the addition is successful, click **Manage application**.
 - h. On the business-level application settings page, click **Save**.
The asset name and type displays in the list of deployed assets.
4. Start the business-level application.
 - a. Click **Applications > Application Types > Business-level applications**.
 - b. On the Business-level applications page, select the check box beside your application.
 - c. Click **Start**.
When the business-level application is running, a green arrow displays for **Status**. If the business-level application does not start, ensure that the deployment target to which the application maps is running and try starting the application again.

What to do next

You can add other assets to your business-level application.

Starting business-level applications

You can start a business-level application that is not running (has a status of Stopped). The application must contain code that can run on a server to start.

Before you begin

The application must be installed on a server. By default, the application starts automatically when the server starts.

About this task

You can start and stop business-level applications manually using the administrative console or wsadmin commands.

This topic describes how to use the administrative console to start a business-level application.

Procedure

1. Go to the Business-level applications page.
Click **Applications > Application Types > Business-level applications** in the console navigation tree.
2. Select the check box for the application you want started.
3. Click **Start**. The product runs the application and changes the state of the application to Started. The status is changed to partially started if not all servers on which the application is deployed are running.

Results

A message stating that the application started displays at the top the page.

If the business-level application does not start, ensure that the deployment target to which the application maps is running and try starting the application again.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using SystemOut.log , SystemErr.log, trace.log, and activity.log files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

If the application contains Service Component Architecture (SCA) composites and does not start, check for the following problems:

- If SCA composite assets do not start, ensure that each asset is mapped to a Version 8 deployment target or to a Version 7 deployment target that supports SCA composites.
- If an asset composition unit uses an Enterprise JavaBeans (EJB) binding and does not start because it has a non-WebSphere target of "null", delete the asset composition unit and add it again to the business-level application. Specify a target that supports SCA composites when you add the asset to the business-level application. You cannot change the target after deployment.
- If the META-INF/sca-deployables directory has multiple SCA composite files and the application does not start because the product cannot obtain the CompUnitInfoLoader value, place only the file that contains the composite in the META-INF/sca-deployables directory. You can place the other composite files anywhere else within the archive.

In multiple-node environments, synchronize the nodes after you save changes to the target before starting the business-level application.

What to do next

To restart a running application, select the application you want to restart, click **Stop** and then click **Start**.

Stopping business-level applications

You can stop a business-level application that is running and has a status of Started).

Before you begin

The application must be running on a product server.

About this task

You can stop applications manually using the administrative console or wsadmin commands.

This topic describes how to use the administrative console to stop a business-level application.

Procedure

1. Go to the Business-level applications page.
Click **Applications > Application Types > Business-level applications** in the console navigation tree.
2. Select the check box for the application you want stopped.
3. Click **Stop**. The product stops the processing of the application and changes the state of the application to Stopped.

Results

The status of the application changes and a message stating that the application stopped displays at the top the page.

What to do next

To restart a stopped application, select the application you want to restart, and then click **Start**.

Updating business-level applications

You can update business-level applications by deleting or changing composition units, or by mapping composition units to different deployment targets.

Before you begin

Determine the changes that you want to make to your application. Also, determine whether the changed application can run on your deployment targets.

The administrative console Server pages show the versions for deployment targets.

If you want to change a composition unit that contains an enterprise bundle archive (EBA) asset, see [Modifying the configuration of an EBA asset](#).

About this task

Updating consists of adding new composition units to an application, replacing or removing composition units, or mapping composition units to different deployment targets.

You can add an asset or shared library composition unit to multiple business-level applications. However, each composition unit for the same asset must have a unique composition unit name. You can add a business-level application composition unit to more than one business-level application.

This topic describes how to update business-level applications using the administrative console. Alternatively, you can use programming or the wsadmin tool.

Procedure

- Delete composition units from your business-level application.
 1. Go to the business-level application settings page.

Click **Applications > Application Types > Business-level applications > *application_name*** in the console navigation tree.

2. Select each composition unit of the application that you want to delete.
 3. Click **Delete**.
 4. On the Delete composition unit from business-level application page, confirm the deletion and click **OK**.
- Add new or updated assets, shared libraries, or other business-level applications to your business-level application.
 1. Update asset binary files or shared libraries as needed.
 2. If you are adding new assets that are not registered with the product management domain, import the assets.
 3. If you are updating existing assets, use the **Update** option to update asset files.

Before updating a shared library, you must manually stop all Java EE applications that depend on that shared library. After updating the shared library, manually restart the Java EE applications. You might need to synchronize configuration changes to the target node. Java EE applications do not automatically restart when a shared library is updated.
 4. On the business-level application settings page, specify the type of composition unit to add.
 - To add an asset, under **Deployed assets**, click **Add > Add Asset**.
 - To add a shared library, under **Deployed assets**, click **Add > Add Shared Library**.
 - To add a business-level application, under **Business-level applications**, click **Add**.
 5. On the New composition unit page, select a unit from the list of available units, and then click **Continue**.
 6. On the Set options page, change the composition unit settings as needed, and then click **Next**.
 7. On the Map composition unit to a target page, change the deployment target as needed, and then click **Next**.

This page is not shown when you add a business-level application.
 8. On the Summary page, click **Finish**.
 9. If the product adds the unit successfully, click **Manage application**.

If the unit addition is not successful, read the messages, and try adding the unit again. Correct the errors noted in any messages.
 10. On the Adding composition unit to the business-level application page, click **Save**.
 11. Repeat these steps to add any other assets, shared libraries, or applications needed by the business-level application.

The business-level application settings page displays the configuration unit names.

- Map composition units to different deployment targets.
 1. On the composition unit settings page, select the composition unit that you want to change.
 2. Under **Current targets**, click **Modify Target**.
 3. On the Map targets page, change the target.
 - a. From the list of available clusters and servers, select a different deployment target.
 - b. Click **>>** to add the deployment target to the **Selected** list.
 - c. To remove a deployment target from the **Selected** list, select the target and click **<<**.
 - d. Click **OK**.

The business-level application settings page displays the selected deployment target.

What to do next

Save the changes to your administrative configuration.

When saving the configuration, synchronize the configuration with the nodes where the application is expected to run.

Deleting business-level applications

After an application no longer is needed, you can delete it.

About this task

Deleting a business-level application removes the application from the product configuration repository and it deletes the application binaries from the file system of all nodes where the application files are installed.

Procedure

1. Go to the Business-level applications page.
Click **Applications > Application Types > Business-level applications** in the console navigation tree.
2. If you need to retain a copy of the application, back up composition units of the application.
3. Delete composition units of the application.
 - a. On the Business-level applications page, click the name of the business-level application that you want to delete.
 - b. On the business-level application settings page, delete each composition unit of the application. Deployed assets and business-level applications can be composition units of a business-level application.
Select one or more composition units and click **Delete**.
 - c. On the Delete composition unit from Business-level application page, confirm the deletion and click **OK**.
 - d. Repeat steps b and c until the business-level application that you want to delete has no more composition units.

Deleting a composition unit removes the configuration from the *profile_root/config/cells/cell_name/cus* directory.

4. Delete the business-level application.
 - a. Select the application that you want to delete.
 - b. Click **Delete**.

Unless the application is used by another business-level application, deleting a business-level application removes the configuration from the *profile_root/config/cells/cell_name/bias* directory.

5. On the Delete business-level application page, confirm the deletion and click **OK**.
6. Save changes made to the administrative configuration.

Results

On single-server products, application binaries are deleted after you save the changes.

On multiple-server products, application binaries are deleted when configuration changes on the deployment manager synchronize with configurations for individual nodes.

What to do next

If using the administrative console **Delete** options does not fully delete a business-level application or its composition units, you can delete the business-level application and its composition units manually from a deployment manager or stand-alone server. Suppose you want to delete a business-level application

named ExampleBLA, and ExampleBLA is not used by another business-level application. Complete the following steps to manually delete the ExampleBLA configurations from the blas and cus directories:

1. Delete the *profile_root/config/cells/cell_name/blas/ExampleBLA* directory.
2. Delete the *profile_root/config/cells/cell_name/cus/ExampleBLA* directory.
3. Save changes made to the administrative configuration.
4. On multiple-server products, synchronize the deployment manager with node configurations.

Chapter 14. Troubleshooting deployment

When you are having problems deploying an application, perform some basic diagnostics and verify your system configuration to solve the problem.

Before you begin

Try to install your application on a product server or cluster. Ensure that your application can be installed to the deployment target. For example, if your application contains modules that support Java Platform, Enterprise Edition (Java EE) 6 or use a Version 8 product feature or API, you must install the application to a Version 8 deployment target.

About this task

Determine which of the following steps apply to the deployment problem and read the suggested topics.

Procedure

- If you cannot install the application, troubleshoot problems deploying applications.
See the topics on application deployment problems and troubleshooting tips.
- If you can install the application but it does not start, troubleshoot problems starting applications.
See the topics on application deployment and startup problems.
- If your application contains many classes with annotations and takes a long time to deploy, reduce annotation searches to speed up deployment.
See the topic on reducing annotation searches during application deployment.
- If you cannot uninstall the application, see the topic on application uninstallation problems.

What to do next

If the topics in this information center do not resolve the deployment problem, examine current information available from IBM Support on known problems and their resolution. IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see topic on troubleshooting help from IBM.

Application deployment problems

You might encounter problems when deploying, installing, or promoting applications. This topic suggests ways to resolve the problems.

What kind of problem are you having?

- “Application does not display ” on page 240
- “Unable to save a deployed application” on page 240
- “WASX7015E error running wsadmin command \$AdminApp installInteractive or \$AdminApp install” on page 241
- “Cannot install a CMP or BMP entity bean in an EJB 3.0 module” on page 241
- “Data definition language (DDL) generated by an assembly tool throws SQL error on target platform” on page 241
- “ADMA0004E: Validation failed” on page 242
- “Cannot load resource WEB-INF/ibm-web-bnd.xmi in archive file” on page 242
- “No valid target is specified in ObjectName anObject for module module_name from installation ” on page 243

- “addNode -includeapps option does not appear to upload all applications to the deployment manager” on page 243
- “Timeout!!!error displays when attempting to install an enterprise application in the administrative console ” on page 243
- “NameNotFoundException message when deploying an application that contains an EJB module” on page 244
- “Compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier” on page 244
- “While uploading documents, addNode -includeapps fails with an OutOfMemoryError exception” on page 244
- “OutOfMemory exception in the deployment manager” on page 244
- “After installing the application onto a different machine, the application does not run” on page 244
- “A single file replaces all application files during application update” on page 245

Check the following first:

- Verify that the logical name that you have specified to appear on the console for your application, enterprise bean module or other resource does not contain invalid characters such as these: - / \ : * ? " < > |.
- If the application was installed using the wsadmin **\$AdminApp install** command with the **-local** flag, restart the server or rerun the command without the **-local** flag.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, check to see if the problem is identified and documented.

Application does not display

Application installed using the wsadmin tool, but the application does not display under **Applications > Application Types > WebSphere Enterprise Applications**.

The application might be installed but you have not saved the configuration:

1. Verify that the application subdirectory is located under the *app_server_root/installedApps* directory.
2. Run the **\$AdminApp list** command and verify that the application is not among those displayed.
 - In the bin directory, run the wsadmin.bat or wsadmin.sh command.
 - From the wsadmin prompt, enter **\$AdminApp list** and verify that the problem application is not among the items that display.
3. Reinstall your application using the wsadmin tool. Run the **\$AdminConfig save** command in the wsadmin tool before exiting.

Unable to save a deployed application

If you are unable to save a deployed application, the problem might be that too many files are opened, exceeding the limit of the operating system.

Only root has authority to adjust the maximum number of files for each process. Complete the following steps to modify the application to close files with disciplines:

1. After you open a file and complete your work, call the close method of the file to release the file handle back to the operating system.
2. Using the java.io.FileInputStream and the FileOutputStream classes as examples, you can invoke their close method to release any system resources that are associated with the stream.

WASX7015E error running wsadmin command \$AdminApp installInteractive or \$AdminApp install

This problem has two possible causes:

- If the full text of the error is similar to:

```
IBM i
WASX7015E: Exception running command:
"$AdminApp installInteractive C:/Documents and Settings/
myUserName/Desktop/MyApp/myapp.ear";
exception information:
com.ibm.bsf.BSFException: error while
evaluating Jacl expression: can't find method "installInteractive"
with 3 argument(s) for class
"com.ibm.ws.scripting.AdminAppClient"
```

The file and path name are incorrectly specified. In this case, since the path included spaces, it was interpreted as multiple parameters by the wsadmin program.

Enter the path of the .ear file correctly. In this case, by enclosing it in double quotes:

```
IBM i
$AdminApp installInteractive "C:\Documents
and Settings\myUserName\Desktop\MyApps\myapp.ear"
```

- If the full text of the error is similar to:

```
IBM i
WASX7015E: Exception running command: "$AdminApp installInteractive c:\MyApps\myapp.ear ";
exception information: com.ibm.ws.scripting.ScriptingException: WASX7115E:
Cannot read input file
"c:\WebSphere\AppServer\bin\MyAppsmypapp.ear"
```

The application path is incorrectly specified. In this case, you must use "forward-slash" (/) separators in the path.

Cannot install a CMP or BMP entity bean in an EJB 3.0 module

When installing an EJB 3.0 module that contains a container-managed persistence (CMP) or bean-managed persistence (BMP) entity bean, the installation fails.

The product does not support installation of applications that have a CMP or BMP entity bean packaged in an EJB 3.0 module. You must package CMP or BMP entity beans in an EJB 2.1 or earlier module.

To resolve this problem:

1. Package the CMP or BMP entity beans in EJB 2.1 or earlier modules.
2. Try installing your application with the EJB 2.1 or earlier modules.

Data definition language (DDL) generated by an assembly tool throws SQL error on target platform

If you receive SQL errors in attempting to execute data definition language (DDL) statements generated by an assembly tool on a different platform, for example if you are deploying a container-managed persistence (CMP) enterprise bean designed on Windows onto a UNIX operating system server, try the following actions:

- Browse the DDL statements for dependencies on specific user identifiers and passwords, and correct as necessary.
- Browse the DDL statements for dependencies on specific server names, and correct as necessary.
- Refer to the message reference of the vendor for causes and suggested actions regarding specific SQL errors. For IBM DB2, you can view the message references online.

If you receive the following error after executing a DDL file created on the Windows operating system or on operating systems such as AIX® or Linux, the problem might come from a difference in file formats:

```
SQL0104N  An unexpected token "CREATE TABLE AGENT (COMM DOUBLE, PERCENT DOUBLE, P"
was found following " ".  Expected tokens may include: " ".
SQLSTATE=42601
```

To resolve this problem:

-  Use EDTF to edit the file.

ADMA0004E: Validation failed

If you see the following error when trying to install an application through the administrative console or the wsadmin command prompt:

```
AppDeploymentException: [ADMA0014E: Validation failed.
ADMA0004E: Validation error in task Specifying the Default Datasource for
EJB Modules JNDI name is not
specified for module beanameBean Jar with URI filename.jar,META-INF/ejb-jar.xml.
You have not specified the
data source for each CMP bean belonging to this module. Either specify the data
source for each CMP beans or
specify the default data source for the entire module.]
```

one possible cause is that, in WebSphere Application Server Version 4.0, it was mandatory to have a data source defined for each CMP bean in each JAR. In Version 5.0 and later releases, you can specify either a data source for a container-managed persistence (CMP) bean or a default data source for all CMP beans in the JAR file. Thus during installation interaction, such as the installation wizard in the administrative console, the data source fields are optional, but the validation performed at the end of the installation checks to see that at least one data source is specified.

To correct this problem, step through the installation again, and specify either a default data source or a data source for each CMP-type enterprise bean.

If you are using the wsadmin tool, use the \$AdminApp installInteractive *filename* command to receive prompts for data sources during installation, or to provide them in a response file.

-  Specify data sources as an option to the \$AdminApp install command.

Cannot load resource WEB-INF/ibm-web-bnd.xml in archive file

The web application tmp.war installs on WebSphere Application Server Versions 5.0 and 5.1, but fails on a WebSphere Application Server Version 6.0 or later server. The application fails to install because the WEB-INF/ibm-web-bnd.xml file contains xml tags that the underlying WCCM model no longer recognizes.

The following error messages display:

```
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xml" in archive "tmp.war"
[2/24/05 14:53:10:297 CST] 000000bc SystemErr      R
AppDeploymentException:
com.ibm.etools.j2ee.commonarchivecore.exception.ResourceLoadException:
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xml" in archive "tmp.war"
[2/24/05 14:53:10:297 CST] 000000bc SystemErr      R
com.ibm.etools.j2ee.commonarchivecore.exception.ResourceLoadException:
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xml" in archive "tmp.war"
!Stack_trace_of_nested_exce!
com.ibm.etools.j2ee.exception.WrappedRuntimeException: Exception occurred loading
WEB-INF/ibm-web-bnd.xml
!Stack_trace_of_nested_exce!
```


To work around this problem, remove the `xmi:type=EJBLocalRef` tag from the `ibm-web-bnd.xml` file. Removing this tag does not affect the application because the tag was previously used for matching the cross document reference type. The application now works for the WebSphere Application Server Version 5.1 and later releases.

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be `.xmi`.
- For an application or module that uses Java EE 5 or later, the file extension must be `.xml`. If `.xmi` files are included with the application or module, the product ignores the `.xmi` files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the `.xmi` file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xml`, `ibm-webservicesclient-bnd.xml`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the `.xmi` file extensions.

No valid target is specified in ObjectName *anObject* for module *module_name* from installation

This error can occur in a clustered environment if the target cell, node, server or cluster into which the application is to be installed is incorrectly specified. For example, it can occur if the target is misspelled.

To correct this problem, check the target names against the actual WebSphere Application Server topology and reenter them with corrections.

addNode -includeapps option does not appear to upload all applications to the deployment manager

This error can occur when some or all applications on the target node are already uploaded to the deployment manager. The `addNode` program detects which applications are already installed and does not upload them again.

Use the administrative console to browse the deployment manager configuration and see the applications that are already installed.

“Timeout!!!” error displays when attempting to install an enterprise application in the administrative console

This error can occur if you attempt to install an enterprise application that has not been deployed.

To correct this problem:

- Open the `file_name.ear` file in an assembly tool and then click **Deploy**. This action creates a file with a name like `Deployed_file_name.ear`.
- In the administrative console, install the deployed EAR file.

NameNotFoundException message when deploying an application that contains an EJB module

If you specify that the EJB deployment tool be run during application installation and the installation fails with a NameNotFoundException message, ensure that the input JAR or EAR file does not contain source files. If there are source files in the input JAR or EAR file, the EJB deployment tools runs a rebuild before generating the deployment code.

To work around this problem, either remove the source files or include all dependent classes and resource files on the class path. Otherwise, the source files or the lack of access to dependent classes and resource files might cause problems during rebuilding of your application on the server.

Compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier

When installing an old application that uses EJB modules that were built to run on WebSphere Application Server Version 5.x or earlier, compilation errors result and EJB deploy fails. The EJB JAR file contains Java source for the old generated code. The old Java source was generated for Version 5.x or before but, when deployed to a WebSphere Application Server Version 6.x or later product, it is compiled using the Version 6.0 or later runtime JAR files.

To work around this problem, remove all .java files from the application EAR file. After the Java source files are removed, you can deploy the application onto a server successfully.

While uploading documents, addNode -includeapps fails with an OutOfMemoryError exception

This error can occur when you use `addNode -includeapps` while you are installing applications with large EAR files. To correct this problem:

- If you are using `addNode` to add a node from the base server, modify the `addNode` script to include the following parameter:
`-Xmxsize`
- If you are adding a node from the administrative console, increase the *maximumHeapSize* in the Java virtual machine settings of the deployment manager, then restart the deployment manager.

OutOfMemory exception in the deployment manager

If you receive an OutOfMemory exception when trying to install applications with large EAR files, try increasing the maximum heap size of the deployment manager.

Check the options you specified on the Java virtual machine page of the administrative console. Increase the *maximumHeapSize* in the Java virtual machine settings of the deployment manager. Then, restart the deployment manager, and try installing the application again.

After installing the application onto a different machine, the application does not run

If your application uses application level resources, its application level node information must be correct for the application to run as expected.

When you add application level resources to an application and deploy the application onto a machine, ensure that the application level node information is correct. Otherwise, when you install the application onto a different machine, it is installed to the wrong location and the application does not run as expected.

You can update the application level node information using an assembly tool. Update the nodeName from deploymentTargets of the deployment.xml file under ibmconfig. Also, ensure that binariesURL from deployedObject of the deployment.xml file has the correct path.

A single file replaces all application files during application update

If you select the **Replace or add a single file** option of the application update wizard and the currently deployed application consists of several files, specify the full path name of the file to be replaced or added for **Specify the path beginning with the installed application archive file to the file to be replaced or added**.

A full path name usually has the structure *directory_path/file_name* and resembles the following:

```
PriceChangeSession.jar/priceChangeSession/priceChangeSessionBean.class
```

Do not specify less than the full path name for **Specify the path beginning with the installed application archive file to the file to be replaced or added**. For example, do not specify only a directory path:

```
PriceChangeSession.jar/priceChangeSession
```

If you specify less than a full path name, all files in the directory of the currently deployed application might be replaced by the single new file that was specified under **Specify the path to the file**.

Application deployment troubleshooting tips

When you first test or run a deployed application, you might encounter problems.

Select the problem you are having with testing or the first run of deployed code for WebSphere Application Server:

- **IBM i** “Application startup problems” on page 251.
- “Web resource is not displayed” on page 255.
- **IBM i** “A client program does not work” on page 254.

You can use the following administrative console pages to inspect the configuration of your applications and JMS resources:

- For a view of the JMS resources for a given application, see the following page: `ae/AppToSIBRefs_DetailForm.dita`.
- For a view of the applications and JMS resources for a given default messaging provider destination, see the following page: `ae/AppsFromSIBRefs_DetailForm.dita`.

IBM i If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM.

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

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

Application startup errors

Use this information for troubleshooting problems that occur when starting an application.

What kind of error do you see when you start an application?

- “HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server”
- “File serving problems” on page 247
- “Graphics do not appear in the JSP file or servlet output” on page 247
- “SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file” on page 248
- **IBM i** “After modifying and saving a JSP file, the change does not show up in the browser ” on page 249
- “Error message: /jspname.jsp(9,0) Include: Mandatory attribute page missing” on page 249
- **IBM i** “The Java source generated from a JSP file is not retained in the temp directory ” on page 249
- “Error Enterprise Application [application name you typed in] not found” on page 250
- “Translation problem with non-English browser input” on page 250
- “Scroll bars do not appear around items in the browser window” on page 250
- “Error Page cannot be displayed, server not found or DNS error” on page 250

The following note applies to the `ibm-web-ext.xmi` references throughout this topic:

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

- For an application or module that uses a Java EE version prior to version 5, the file extension must be `.xmi`.
- For an application or module that uses Java EE 5 or later, the file extension must be `.xml`. If `.xmi` files are included with the application or module, the product ignores the `.xmi` files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the `.xmi` file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the `.xmi` file extensions.

HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server

If your HTTP server appears to be functioning correctly, and the Application Server also works on its own, but browser requests sent to the HTTP server for pages are not being served, a problem exists in the WebSphere Application Server plug-in.

In this case:

1. Determine whether the HTTP server is attempting to serve the requested resource itself, rather than forwarding it to the WebSphere Application Server.
 - a. Browse the HTTP server access log (*IHS install root/logs/access.log* for IBM HTTP Server). It might indicate that it could not find the file in its own document root directory.
 - b. Browse the plug-in log file as described below.
2. Refresh the `plugin-cfg.xml` file that determines which requests sent to the HTTP server are forwarded to the WebSphere Application Server, and to which Application Server.

Use the console to refresh this file:

- In the WebSphere Application Server administrative console, expand the Environment tree control.
- Click Update WebSphere Plugin.
- Stop and restart the HTTP server.

IBM i If you are using IBM HTTP Server for iSeries® or Lotus Domino® for iSeries, you do not need to restart the HTTP server.

- Retry the web request.

IBM i If you have created a web server definition to model your web server instance, the file is located under *profile_root/config/cells/cell_name/nodes/Web_server_node_name/servers/Web_server_name*. If you have not, the file is located under *profile_root/config/cells*.

3. Browse the *plugin_install_root/logs/web_server_name/http_plugin.log* file for clues to the problem. Make sure the timestamps with the most recent plug-in information stanza, which is printed out when the plug-in is loaded, correspond to the time the web server started.
4. Turn on plug-in tracing by setting the `LogLevel` attribute in the *plugin-cfg.xml* file to `Trace` and reloading the request. Browse the *plugin_install_root/logs/Web_server_name/http_plugin.log* file. You should be able to see the plug-in attempting to match the request URI with the various URI definitions for the routes in the *plugin-cfg.xml*. Check which rules the plug-in is not matching against and then figure out if you need to add additional ones. If you just recently installed the application you might need to manually regenerate the plug-in configuration to pick up the new URIs related to the new application.

For further details on troubleshooting plug-in-related problems, see *Webserver plug-in troubleshooting tips* located in the *Administering applications and their environment* PDF book.

File serving problems

If text output appears on your JSP- or servlet-supported web page, but image files do not:

- Verify that your files are in the right place: the **document root** directory of your web application. WebSphere Application Server follows the J2EE standard, which means that the document root is the *web_module_name.war* directory of your deployed web application.

IBM i Typically this directory will be found in the *profile_root/installedApps/nodename/appname.ear* directory or *profile_root/installedApps/nodename/appnameNetwork.ear* directory.

If the files are in a subdirectory of the document root, verify that the reference to the file reflects that. That is, if the *invoices.html* file is stored in Windows directory *web_module_name.war\invoices*, then links from other pages in the web application to display it should read "*invoices\invoices.html*", not "*invoices.html*".

- Verify that your web application is configured to enable file serving (in other words, that it is enabled to display static resources like image and .html files):
 1. View the file serving property of the hosting web module by browsing the source .war file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
 2. Edit the `fileServingEnabled` property in the deployed web application *ibm-web-ext.xmi* configuration file.

IBM i The file typically is found in the *profile_root/config/cells/nodename* or *nodenameNetwork/applications/application_name/deployments/application_name/Web_module_name/web-inf* directory.

Graphics do not appear in the JSP file or servlet output

If text output appears on your JSP- or -servlet-supported web page, but image files do not:

- Verify that your graphic files are in the right place: the document root directory of your web application. The product follows the J2EE standard, which means that the document root is the *web_module_name.war* directory of your deployed web application.

IBM i Typically, this directory is found in the *profile_root/installedApps/nodename/appname.ear* directory or *profile_root/installedApps/nodename/appnameNetwork.ear* directory.

If the graphics files are in a subdirectory of the document root, verify that the reference to the graphic reflects that; for example, if the banner.gif file is stored in Windows directory *web_module_name.war/images*, the tag to display it should read: ``, not ``.

- Verify that your web application is configured to enable file serving (that is, display of static resources like image and .html files).
 1. View the file serving property of the hosting web module by browsing the source .war file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
 2. Edit the **fileServingEnabled** property in the deployed web application `ibm-web-ext.xmi` configuration file.

IBM i The file typically is found in the *profile_root/config/cells/nodename* or *nodenameNetwork/applications/application_name/deployments/application_name/Web_module_name/web-inf* directory.
 3. After completing the previous step:
 - In the administrative console, expand the Environment tree control .
 - Click **Update WebSphere Plugin**.
 - Stop and restart the HTTP server and retry the web request.

SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file

If this error appears in a browser when trying to access a new or modified .jsp file for the first time, the most likely cause is that the JSP file Java source failed (was incorrect) during the `javac` compilation phase.

IBM i Check the SystemErr.log file for a compiler error message, such as:

```
C:\WASROOT\temp\ ... test.war\_myJsp.java:14: \Duplicate variable declaration: int myInt was int myInt
int myInt = 122;
String myString = "number is 122";
static int myStaticInt=22;
int myInt=121;
    ^
```

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Fix the problem in the JSP source file, save the source and request the JSP file again.

If this error occurs when trying to serve a JSP file that was copied from another system where it ran successfully, then there is something different about the new server environment that prevents the JSP file from running. Browse the text of the error for a statement like:

```
Undefined variable or class name: MyClass
```

This error indicates that a supporting class or jar file is not copied to the target server, or is not on the class path. Find the `MyClass.class` file, and place it on the web module `WEB-INF/classes` directory, or place its containing .jar file in the Web module `WEB-INF/lib` directory.

Verify that the URL used to access the resource is correct by doing the following:

- For a JSP file, html file, or image file: `http://host_name/Web_module_context_root/subdir under doc root, if any/filename.ext`. The document root for a web application is the *application_name.WAR* directory of the installed application.

- For example, to access the myJsp.jsp file, located in c:\WebSphere\ApplicationServer\installedApps\myEntApp.ear\myWebApp.war\invoices on myhost.mydomain.com, and assuming the context root for the myWebApp web module is myApp, the URL is http://myhost.mydomain.com/myApp/invoices/myJsp.jsp.
- JSP serving is enabled by default. File serving for HTML and image files must be enabled as a property of the web module, in an assembly tool, or by setting the **fileServingEnabled** property to **true** in the `ibm-web-ext.xmi` file of the installed web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
- For servlets served by class name, the URL is `http://hostname/Web_module_context_root/servlet/packageName.className`.
 - IBM i** For example, to access `myCom.myServlet.class`, located in `profile_root/installedApps/myEntApp.ear/myWebApp.war/WEB-INF/classes`, and assuming the context root for the myWebApp module is “myApp”, the URL would be `http://myhost.mydomain.com/myApp/servlet/myCom.MyServlet`.
- Serving servlets by class name must be enabled as a property of the web module, and is enabled by default. File serving for HTML and image files must be enabled as a property of the Web application, in an assembly tool, or by setting the **fileServingEnabled** property to **true** in the `ibm-web-ext.xmi` file of the installed web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Correct the URL in the “from” HTML file, servlet or JSP file. An HREF with no leading slash (/) inherits the calling resource context. For example:

- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `ServletB` resolves to `"http://hostname/myapp/servlet/ServletB"`
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `"servlet/ServletB"` resolves to `"http://hostname/myapp/servlet/servlet/ServletB"` (an error)
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `"/ServletB"` resolves to `"http://hostname/ServletB"` (an error, if `ServletB` requires the same context root as `MyServlet`)

IBM i After modifying and saving a JSP file, the change does not show up in the browser

It is probable that the web application is not configured for servlet reloading, or the reload interval is too high.

To correct this problem, in an assembly tool, check the Reloading Enabled flag and the Reload Interval value in the IBM Extensions for the web module in question. Enable reloading, or if it is already enabled, then set the Reload Interval lower. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Error message: /jspname.jsp(9,0) Include: Mandatory attribute page missing

The error “Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing” appears when attempting to browse JSP file

It is probable that the JSP file failed during the translation to Java phase. Specifically, a JSP directive, in this case an Include statement, was incorrect or referred to a file that could not be found.

To correct this problem, fix the problem in the JSP source, save the source and request the JSP file again.

IBM i The Java source generated from a JSP file is not retained in the temp directory

It is probable that the JSP processor is not configured to keep generated Java source.

In an assembly tool, check the JSP Attributes under Assembly Property Extensions for the web module in question. Make sure the **keepgenerated** attribute is there and is set to true. If not, set this attribute and restart the web application. To see the results of this operation, delete the class file from the temp directory to force the JSP processor to translate the JSP source into Java source again. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Error “Enterprise Application [application name you typed in] not found”

The JSP Batch Compiler fails with the message “Enterprise Application [application name you typed in] not found”

It is probable that the full enterprise application path and name, starting with the .ear subdirectory that resides in the applications directory is expected as an argument to the JspBatchCompiler tool, not just the display name.

IBM i The directory path is *profile_root/config/cells/node_nameNetwork/applications*.

For example:

- "JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp" is correct, as opposed to
- "JspBatchCompiler -enterpriseapp.name sampleApp", which is incorrect.

Translation problem with non-English browser input

If non-English-character-set browser input cannot be translated after being read by a servlet or JSP file, ensure that the request parameters are encoded according to the expected character set before reading. For example, if the site is Chinese, the target .jsp file should have a line:

```
req.setCharacterEncoding("gb2312");
```

before any req.getParameter method calls.

This problem affects servlets and jsp files ported from earlier versions of WebSphere Application Server, which converted characters automatically based upon the locale of the WebSphere Application Server.

Scroll bars do not appear around items in the browser window

In some browsers, tree or list type items that extend beyond their allotted windows do not have scroll bars to permit viewing of the entire list.

To correct this problem, right-click on the browser window and click **Reload** from the menu.

Error “Page cannot be displayed, server not found or DNS error”

Error “Page cannot be displayed, server not found or DNS error” appears when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer

This error can occur when an HTTP timeout causes the servant to be brought down and restarted. To correct this problem, increase the ConnectionIOTimeout value:

1. From the administrative console, select **System administration Deployment manager Administration Services Custom Properties**
2. Select ConnectionIOTimeout.
3. Increase the ConnectionIOTimeout value.
4. Click **OK**.

Application startup problems

When an application is not starting or starting with errors, the problem could be from one of various sources.

What kind of error do you see when you start an application?

- “WSVR0100W: An error occurred initializing, application_name java.lang.NullPointerException when starting a migrated application”
- A “java.lang.ClassNotFoundException: classname Bean_AdderServiceHome_04f0e027Bean” error occurs
- A “ConnectionFactory E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter” on page 252 error occurs
- “NMSV0605E: A Reference object looked up from the context... error when starting an application” on page 253.
- “A Page Not Found, Array Index Out of Bounds, or other error when an updated application restarts” on page 253

If none of these errors match the error you see:

- Browse the log files of the application server for this application looking for clues. By default, these files are: *app_server_root/logs/server_name/SystemErr.log* and *SystemOut.log*.
- Look up any error or warning messages in the message reference table by clicking the **Reference** view and expanding **Messages**.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using *SystemOut.log*, *SystemErr.log*, *trace.log*, and *activity.log* files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see IBM Support troubleshooting information.

WSVR0100W: An error occurred initializing, *application_name* java.lang.NullPointerException when starting a migrated application

After you migrate an enterprise application to Version 8.0, the application might not start. Attempts to start the application result in an error such as WSVR0100W: An error occurred initializing, *application_name* java.lang.NullPointerException.

Examine the *deployment.xml* file of the migrated application, and remove *targetMapping* statements such as the following:

```
<targetMappings xmi:id="DeploymentTargetMapping_1279594183813" enable="true"/>
```

Then, try starting the application again. The Version 8.0 runtime has an application validation process that might not support migrated *targetMappings* settings.

java.lang.ClassNotFoundException: *classname* Bean_AdderServiceHome_04f0e027Bean

An similar exception occurs when you try to start an undeployed application containing enterprise beans, or containing undeployed enterprise bean modules.

IBM i Enterprise JavaBeans modules created in an assembly tool intentionally have incomplete configuration information. Deploying these modules completes the configuration by reading the module's deployment descriptor and completing platform- or installation-dependent settings and adding related classes to the Enterprise JavaBeans JAR file.

To avoid this problem, do the following:

- Use an assembly tool and administrative console to generate deployment code and install the application or Enterprise JavaBeans module onto a server.
 1. Uninstall the application or Enterprise JavaBeans module in the administrative console.
 2. Configure your assembly tool so the target server is a WebSphere Application Server installation. If you do not have access to the target server, you can specify a false location such as /temp. Specifying a false location enables you to assemble and generate deployment code for the enterprise bean.
 3. In the Project Explorer view of an assembly tool, right-click the enterprise bean (Enterprise JavaBeans) in the undeployed .ear file containing the Enterprise JavaBeans module or the stand-alone undeployed Enterprise JavaBeans JAR file, and click **Deploy**. If your assembly tool can access the WebSphere Application Server target server, deployment code is generated for the Enterprise JavaBeans and the assembly tool attempts to install the application or module onto the target server. If your assembly tool cannot access the WebSphere Application Server target server or the installation fails, use the deployment code that is generated for the next step.
For information on using an assembly tool, refer to the topic on assembling applications.
 4. Use the `wsadmin $AdminApp install` command or the administrative console to install the deployed version created by the assembly tool.
- If you use the `wsadmin $AdminApp install` command, uninstall it and then reinstall using the `-EJBDeploy` option. Follow the `install` command with the `$AdminConfig save` command.

ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter

This error occurs when an enterprise bean developed to the Enterprise JavaBeans 1.1 specification is deployed with a WebSphere Application Server V5 J2C-compliant data source, which is the default data source. By default, persistent enterprise beans created under WebSphere Application Server V4.0 using the Application Assembly Tool fulfill the Enterprise JavaBeans 1.1 specification. To run on WebSphere Application Server V6, these enterprise beans must be associated with a WebSphere Application Server V4.0-type data source.

Either modify the mapping in the application of enterprise beans to associate 1.x container managed persistence (CMP) beans to associate them with a V4.0 data source or delete the existing data source and create a V4.0 data source with the same name.

To modify the mapping in the application of enterprise beans, in the WebSphere Application Server administrative console, select the properties for the problem application and use **Map resource references to resources** or **Map data sources for all 1.x CMP beans** to switch the data source the enterprise bean uses. Save the configuration and restart the application.

To delete the existing data source and create a V4.0 data source with the same name:

1. In the administrative console, click **Resources > Manage JDBC Providers > JDBC_provider_name > Data sources**.
2. Delete the data source associated with the Enterprise JavaBeans 1.1 module.
3. Click **Resources > Manage JDBC Providers > JDBC_provider_name > Data sources (Version 4)**.
4. Create the data source for the Enterprise JavaBeans 1.1 module.
5. Save the configuration and restart the application.

NMSV0605E: “A Reference object looked up from the context...” error when starting an application

If the full text of the error is similar to:

```
[7/17/02 15:20:52:093 CDT] 5ae5a5e2 Ur1ContextHel W NMSV0605E:
A Reference object looked up from the context
"java": with the name "comp/PM/WebSphereCMPConnectionFactory" was sent to the JNDI Naming Manager
and an exception resulted. Reference data follows:
Reference Factory Class Name: com.ibm.ws.naming.util.IndirectJndiLookupObjectFactory
Reference Factory Class Location URLs:
Reference Class Name: java.lang.Object
Type: JndiLookupInfo
Content: JndiLookupInfo: ; jndiName="eis/jdbc/MyDatasource_CMP"; providerURL="";
initialContextFactory=""
```

then the problem might be that the data source intended to support a CMP enterprise bean is not correctly associated with the enterprise bean.

To resolve this problem:

1. Select the **Use this Data Source in container managed persistence (CMP)** check box in the data source “General Properties” panel of the administrative console.
2. Verify the JNDI name in either of the following ways:
 - Verify that the JNDI name given in the administrative console under **Resources > Manage JDBC Providers > DataSource > JNDI Name** for DataSource matches the JNDI name given for CMP or BMP resource bindings at the time of assembling the application in an assembly tool.
 - Check the JNDI name for CMP or BMP resource bindings specified in the code by J2EE application developer. Open the deployed .ear folder in an assembly tool, and look for the JNDI name for your entity beans under CMP or BMP resource bindings. Verify that the names match.

A Page Not Found, Array Index Out of Bounds, or other error when an updated application restarts

If an application is updated while it is running, WebSphere Application Server automatically stops the application or only its changed components, updates the application logic, and restarts the stopped application or its components. For more information on the restarting of updated applications, refer to Fine-grained recycle behavior in *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 5 Flexible options for updating deployed applications*.

A Page Not Found, Array Index Out of Bounds, or other error might occur during restarting.

To minimize the occurrence of such errors, update applications in a test environment before updating the applications in a production environment. Do not put changes directly into a production environment.

Reducing annotation searches during application deployment

Enterprise applications that contain many classes with annotations might take a long time to deploy. Java EE 5 introduced annotations to add metadata to Java classes. Because of performance issues associated with reflection and because classes are not always loadable at deployment, bytecode scanning technology is used to retrieve annotation metadata. Java EE 5 or later applications with many classes might experience long deployment times because every class within the application is inspected during deployment. You can reduce the number of annotations to inspect by specifying the modules and Java packages to ignore for annotations processing in the `amm.filter.properties` file or by configuring system properties.

Before you begin

Install an application that supports Java Platform, Enterprise Edition (Java EE) 5 or later on a product server or cluster. If deployment is unreasonably slow and you will be deploying this application again in the future, complete a procedure in this topic to reduce the number of classes that are searched for annotations during deployment.

About this task

The product provides a configurable filtering function to reduce the number of classes that are searched for annotations. You can identify which modules or Java packages to ignore for annotations processing through two properties:

- Ignore-Scanning-Archives
- Ignore-Scanning-Packages

A default set of values is provided in the `amm.filter.properties` file in `app_server_root/properties`. The property values provide both coarse and fine grained control over the search scope for annotations processing. Use of the Ignore-Scanning-Archives property reduces deployment time more than use of the Ignore-Scanning-Packages property. The syntax for the Ignore-Scanning-Archives and Ignore-Scanning-Packages properties follows the comma-separated value convention. No wildcard or regular expressions are permitted and values are case-sensitive.

The default set of values can be changed by an administrator or augmented by a user using one of the following steps.

Procedure

- Place an `amm.filter.properties` file in the `profile_root/properties` directory.
 - Use system properties to supply values for the Ignore-Scanning-Archives and Ignore-Scanning-Packages properties.
 - The `com.ibm.ws.amm.scan.context.filter.archives` system property supplies values for the Ignore-Scanning-Archives property.
 - The `com.ibm.ws.amm.scan.context.filter.packages` system property supplies values for the Ignore-Scanning-Packages property.
- See the topic on Java virtual machine custom properties.
- Add Ignore-Scanning-Archives and Ignore-Scanning-Packages entries to the application manifest, `META-INF/MANIFEST.MF`.

Note: When updating the application manifest, follow line-length limitations and other constraints for the manifest.

- Add Ignore-Scanning-Archives and Ignore-Scanning-Packages entries to the module manifest.

Note: When updating the module manifest, follow line-length limitations and other constraints for the manifest.

What to do next

Install the application again. If deployment continues to be slow, specify more modules and Java packages to ignore.

A client program does not work

What kind of problem are you seeing?

ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages) or both

A possible cause of this problem is that both IIS for serving Active Server Pages (ASP) files and an HTTP server that supports WebSphere Application Server (such as IBM HTTP Server) are deployed on the same host. This deployment leads to misdirected HTTP traffic if both servers are listening on the same port (such as the default port 80).


To resolve this problem, either:

- Open the IIS administrative panel, and edit the properties of the default web server to change the port number to a value other than 80
- Install IIS and the HTTP server on separate servers.

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

Plants by WebSphere Catalog Manager (pbwsCatalogMgr) exceptions

When you federate a stand-alone server into a Deployment Manager cell, the bootstrap port number of the application server may change. This will cause the client to not be able to communicate with the server, thus causing an exception. The following scenario may cause an exception when you start Plants by WebSphere:

1. Install a stand-alone WebSphere Application Server.
2. Run the Plants by WebSphere example.
3.  Create a Deployment Manager (DMGR) using the Profile Management tool or by using the **manageprofiles** command.
4. Federate the stand-alone WebSphere Application Server into a Deployment Manager cell using the **addNode** command.
5. Start **pbwsCatalogMgr**.

To avoid the exception, locate the new (changed) port number on the server and modify the client configuration to match the port number on the server.

1. Go to *was_server_root\profiles\your_server_name\config\cells\your_cell\nodes\your_node*.
 - a. Open the *serverindex.xml* file.
 - b. Locate the **BOOTSTRAP_ADDRESS** port number of the application server, for example 9810.
2. Assign this port number to the client to communicate with your newly-federated application server. Go to *was_client_root\bin* and edit the *setupClient.bat* file.
3. Locate the line 'SET SERVERPORTNUMBER' and set the value for it to 9810.

If you have security enabled, ensure that the bus security is also enabled and that a user is defined to the bus connector role before running **pbwsCatalogMgr**.
4. Restart the node agent and the application server.

The client is now properly set up to start **pbwsCatalogMgr**.

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

Web resource is not displayed

Use this information to troubleshoot problems that occur when attempting to display a resource in a browser.

If you are not able to display a resource in your browser, follow these steps:

1. Verify that your HTTP server is healthy by accessing the URL `http://server_name` from a browser and seeing whether the Welcome page appears. This action indicates whether the HTTP server is up and running, regardless of the state of WebSphere Application Server.
2. If the HTTP server Welcome page does not appear, that is, if you get a browser message like page cannot be displayed or something similar, try to diagnose your web server problem.
3. If the HTTP server appears to function correctly, the Application Server might not be serving the target resource. Try to access the resource directly through the Application Server instead of through the HTTP server.

If you cannot access the resource directly through the Application Server, verify that the URL used to access the resource is correct.

If the URL is incorrect and it is created as a link from another JavaServer Pages (JSP) file, servlet, or HTML file, try correcting it in the browser URL field and reloading, to confirm that the problem is a malformed URL. Correct the URL in the "from" HTML file, servlet or JSP file.

If the URL appears to be correct, but you cannot access the resource directly through the Application Server, verify the health of the hosting application server and web module:

- a. View the hosting application server and web module in the administrative console to verify that they are up and running.
- b. Copy a simple HTML or JSP file, such as `SimpleJsp.jsp`, which is in the WebSphere Application Server directory structure, to your web module document root, and try to access the file. If successful, the problem is with the resource.

IBM i View the JVM log of your Application Server to find out why your resource cannot be found or served .

4. If you can access the resource directly through the Application Server, but not through an HTTP server, the problem lies with the HTTP plug-in, the component that communicates between the HTTP server and the WebSphere Application Server.
5. If the JSP file and the servlet output are served, but not static resources such as `.html` and image files, see the steps for enabling file serving.
6. If certain resources display correctly, but you cannot display a servlet by its class name:
 - Verify that the servlet is in a directory in the web module class path, such as in the `/web_module_name.war/WEB-INF/classes` directory.
 - Verify that you specify the full class name of the servlet, including its package name, in the URL.
 - Verify that `"/servlet"` precedes the class name in the URL. For example, if the root context of a web module is "myapp", and the servlet is `com.mycom.welcomeServlet`, then the URL reads:
`http://hostname/myapp/servlet/com.mycom.welcomeServlet`
 - Verify that serving the servlets by class name is enabled for the hosting web module by opening the source web module in an assembly tool and browsing the *serve servlets by classname* setting in the IBM Extensions property page. If necessary, enable this flag and redeploy the web module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
 - For servlets or other resources served by mapped URLs, the URL is `http://hostname/Web module context root/mappedURL`.

IBM i If none of these steps fixes your problem, see if the problem has been identified and documented by looking at available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see Troubleshooting help from IBM.

Diagnosing web server problems

If you are unable to view the welcome page of your HTTP server, determine if the server is operating properly.

If the HTTP server does not start:

- Examine the HTTP server error log for clues.

- Try restoring the HTTP server to its configuration prior to installing WebSphere Application Server and restarting it. If you are using IBM HTTP Server:
 - Rename the file `IHS_install_dir\httpd.conf`.
 - Copy the `httpd.conf.default` file to the `httpd.conf` directory.
 - If Apache is running, stop and restart it.
- For the Sun ONE (iPlanet) Web Server, restore the `obj.conf` configuration file for Sun ONE V4.1 and both `obj.conf` and `magnus.conf` files for Sun ONE V6.0 and later.
- For the Microsoft Internet Information Server (IIS), remove the WebSphere Application Server plug-in through the IIS administrative GUI.


If restoring the HTTP server default configuration file works, manually review the configuration file that has WebSphere Application Server updates to verify directory and file names for WebSphere Application Server files. If you cannot manually correct the configuration, you can uninstall and reinstall WebSphere Application Server to create a clean HTTP configuration file.

If restoring the default configuration file does not help, contact technical support for the web server you are using. If you are using IBM HTTP Server with WebSphere Application Server, check available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see Troubleshooting help from IBM.

Accessing a web resource through the application server and bypassing the HTTP server

You can bypass the HTTP server and access a web resource through the application server. It is not recommended to serve a production website in this way, but it provides a good diagnostic tool when it is not clear whether a problem resides in the HTTP server, WebSphere Application Server, or the HTTP plug-in.

To access a web resource through the Application Server:

1. Determine the port of the HTTP service in the target application server.
 - a. In the administrative console, click **Servers > Server Types > WebSphere application servers > application_server > Web container**.
 - b. Under the Additional Properties of the web container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the application server.
 - c.  There can be more than one port listed. In the default application server (server1), for example, 9060 is the port reserved for administrative requests, 9443 and 9043 are used for SSL-encrypted requests. To test the sample "snoop" servlet, for example, use the default application port 9080, unless it changes.
2. Use the HTTP transport port number of the application server to access the resource from a browser. For example, if the port is 9080, the URL is `http://hostname:9080/myAppContext/myJSP.jsp`.
3. If you are still unable to access the resource, verify that the HTTP transport port is in the "Host Alias" list:
 - a. Click **Servers > Server Types > WebSphere application servers > application_server > Web container > HTTP transports** to check the Default virtual host and the HTTP transport ports used by this application server.
 - b. Click **Environment > Virtual hosts > default_host > Host Aliases** to check if the HTTP transport port exists. Add an entry if necessary. For example, if the HTTP port for your application is server is 9080, add a host alias of `*:9082`.

Application uninstallation problems

When you try to uninstall an application or node, you might encounter problems. This topic suggests ways to resolve uninstallation problems.

What kind of problem are you having?

- After uninstalling an application through wsadmin tool, the application continues to run and throws "DocumentIOException"
- The removeNode command does not remove the installed application from the deployment manager
- I cannot display the syntax for the removeNode command.

If none of these steps fixes your problem:

- Make sure that the application and its web and EJB modules are in a stopped state before uninstalling.
- If you are uninstalling or installing an application using wsadmin, make sure that you are using the -conntype NONE option to invoke wsadmin and enable local mode. To use the -conntype NONE option, stop the hosting application server before uninstalling the application.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there, contact IBM support

After uninstalling application through the wsadmin tool, the application throws "DocumentIOException"

If this exception occurs after the application was uninstalled using wsadmin with the -conntype NONE option:

- Restart the server or,
- Rerun the uninstall command without the -conntype NONE option.

The removeNode command does not remove the installed application from the deployment manager

If the applications were installed indirectly using the **addNode** command with the **-includeapps** option, then **removeNode** will not uninstall them, since they may be in use by other nodes. These applications must be explicitly uninstalled, for example through the administrative console.

I cannot display the syntax for the removeNode command

Unlike the **addNode** command, the **removeNode** command is valid with no parameters, so executing it will execute the operation, that is, remove the node, without displaying the command syntax.

To see the valid options for **removeNode**, run **removeNode -?** or **removeNode -help**.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org>. You may obtain a copy of the Apache License at <http://www.apache.org/licenses/LICENSE-2.0>.

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

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

Trademarks and service marks

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

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

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Index

A

- applications
 - installation bindings 136
 - settings
 - installation 135
- assets 210
 - deleting 215
 - exporting 215
 - importing 204
 - managing 209
 - settings 206
 - adding 223
 - updating 213
 - uploading 205
 - updating 211
- authentication
 - JASPI enablement
 - applications 165

B

- backend ID
 - settings 186
- business-level applications 199, 220
 - assets 202
 - composition units 202
 - creating 216, 232
 - administrative console 217
 - deleting 237
 - settings 221, 227
 - starting 233
 - stopping 234
 - updating 235

C

- class loaders 27
 - Java EE 21
 - Java EE applications 28
 - settings 27
 - web modules 30
 - WebSphere server 26
- client modules
 - property settings 155
 - settings 154
- cnfiguration elements
 - jndiEntry 105
- common OSGi bundles
 - sharing 106
- composition units
 - settings 229
 - adding 223
- configuration elements
 - bundleRepository 106
 - context-root 115
- context-root
 - deploying 103

D

- deployment
 - applications 121
 - business-level applications 199
 - EAR files
 - default bindings 133
 - enterprise modules with JSR-88 194
 - Java EE application files
 - WebSphere targets 121
 - Java EE files
 - administrative console 126
 - Java EE modules
 - deployment targets 122, 124

E

- EJB Deploy
 - settings 174
- Enterprise JavaBeans (EJB)
 - JNDI names for beans 156
 - references 159
 - settings
 - binding EJB business settings 157
- entity manager
 - configuring 116
- environment entries
 - settings
 - applications 183
 - client module 181
 - EJB modules 182

I

- installation options
 - settings 142

J

- JDBC applications
 - deploying 111
- JDBC driver location
 - deploying 111
- JDBC drivers
 - deploying 111
- JDBC tracing options 112
- JNDI
 - settings
 - JCA objects 186
- JNDI binding for constants 105
- JPA applications
 - configuring 116

L

- Liberty server
 - packaging 104

M

- messages
 - settings
 - destination references 185
- metadata
 - settings
 - modules 189
- module build ID
 - settings 192
- module customization
 - DConfigBeans 196
- modules
 - settings 152

O

- options
 - relationship options
 - settings 226
 - settings 223
- OSGi applications
 - deploying 103

S

- server features
 - deploying 111

- shared library reference and mapping
 - settings 177
- shared library relationship and mapping
 - settings 178, 222

T

- target mapping
 - settings 225

V

- virtual hosts
 - settings 163

W

- web applications
 - settings
 - initial parameters for servlets 180
- web services
 - samples installation 134
 - settings
 - options to perform web services deployment 191